

MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

INTRODUCTION TO SWIFT

OPEN SOURCE LANGUAGE FOR IOS

MEMORY MANAGEMENT IN SWIFT

OPEN SOURCE

THE NEW BUSINESS ERA

SAMHAIN - FREE

OPEN SOURCE FILE INTEGRITY MONITORING - HIDS

INTERVIEW WITH MICHA MAZAHARI

CEO OF PAW INC.

MODEL VIEW WHATEVER

MVP BY MIKE POTEL

WHY FREEBSD ?

VOL. 10 NO. 02

ISSUE 02/2016(78)

1898-9144

FREENAS MINI STORAGE APPLIANCE

IT SAVES YOUR LIFE.



HOW IMPORTANT IS YOUR DATA?

Years of family photos. Your entire music and movie collection. Office documents you've put hours of work into. Backups for every computer you own. We ask again, *how important is your data?*

NOW IMAGINE LOSING IT ALL

Losing one bit - that's all it takes. One single bit, and your file is gone.

The worst part? **You won't know until you absolutely need that file again.**



Example of one-bit corruption

THE SOLUTION

The FreeNAS Mini has emerged as the clear choice to save your digital life. **No other NAS in its class offers ECC (error correcting code) memory and ZFS bitrot protection to ensure data always reaches disk without corruption and *never degrades over time.***

No other NAS combines the inherent data integrity and security of the ZFS filesystem with fast on-disk encryption. No other NAS provides comparable power and flexibility. The FreeNAS Mini is, hands-down, the best home and small office storage appliance you can buy on the market. **When it comes to saving your important data, there simply is no other solution.**

The Mini boasts these state-of-the-art features:

- 8-core 2.4GHz Intel® Atom™ processor
- Up to 16TB of storage capacity
- 16GB of ECC memory (with the option to upgrade to 32GB)
- 2 x 1 Gigabit network controllers
- Remote management port (IPMI)
- Tool-less design; hot swappable drive trays
- FreeNAS installed and configured



<http://www.iXsystems.com/mini>



FREENAS CERTIFIED STORAGE



With over six million downloads, FreeNAS is undisputedly *the* most popular storage operating system in the world.

Sure, you could build your own FreeNAS system: research every hardware option, order all the parts, wait for everything to ship and arrive, vent at customer service because it *hasn't*, and finally build it yourself while hoping everything fits - only to install the software and discover that the system you spent *days* agonizing over **isn't even compatible**. Or...

MAKE IT EASY ON YOURSELF

As the sponsors and lead developers of the FreeNAS project, iXsystems has combined over 20 years of hardware experience with our FreeNAS expertise to bring you FreeNAS Certified Storage. **We make it easy to enjoy all the benefits of FreeNAS without the headache of building, setting up, configuring, and supporting it yourself.** As one of the leaders in the storage industry, you know that you're getting the best combination of hardware designed for optimal performance with FreeNAS.

Every FreeNAS server we ship is...

- » Custom built and optimized for your use case
- » Installed, configured, tested, and guaranteed to work out of the box
- » Supported by the Silicon Valley team that designed and built it
- » Backed by a 3 years parts and labor limited warranty

As one of the leaders in the storage industry, you know that you're getting the best combination of hardware designed for optimal performance with FreeNAS. **Contact us today for a FREE Risk Elimination Consultation with one of our FreeNAS experts.** Remember, every purchase directly supports the FreeNAS project so we can continue adding features and improvements to the software for years to come. **And really - why would you buy a FreeNAS server from *anyone* else?**



FreeNAS 1U

- Intel® Xeon® Processor E3-1200v2 Family
- Up to 16TB of storage capacity
- 16GB ECC memory (upgradable to 32GB)
- 2 x 10/100/1000 Gigabit Ethernet controllers
- Redundant power supply

FreeNAS 2U

- 2x Intel® Xeon® Processors E5-2600v2 Family
- Up to 48TB of storage capacity
- 32GB ECC memory (upgradable to 128GB)
- 4 x 1GbE Network interface (Onboard) - (Upgradable to 2 x 10 Gigabit Interface)
- Redundant Power Supply

<http://www.ixsystems.com/storage/freenas-certified-storage/>



Dear Readers,

February, the month of love, is behind us. This month we have Easter. We hope you have a great time with your families this month of Easter (as we do every month ;)).

This issue is predominantly focused on iOS software. February was not that favorable for Apple, was it? Stories about Apple clashing with the America's Federal Bureau of investigation (FBI) have been featured everywhere, so our focus this month is a very timely coincidence. While we will not be discussing security, McAfee, Snowden, Trump or any other person trying to pull the spotlight in their direction on this subject, we would like to know what you think about this. Which side are you taking, Apple's or the FBI's?

This issue opens with BSD World Monthly News, as always. Next, you will find a couple of articles about Swift. To begin with, "Introduction to Swift, Open Source Language for iOS" by David Carlier. After that, "Open Source Might Be the New Business Era!" is a very nice and easily read article by Georgios Pessios. It is his debut, let us know what you think about his first article. The topic of Swift closes with "Memory Management in Swift" by Arash Z. Jahangiri.

Now we move on to "Why FreeBSD?" by Hamza Sheikh. Do you agree with his arguments?

Have you ever heard about Samhain? Now you can find out more from the article "Samhain - Free, Open Source File Integrity Monitoring / HIDS" by Leonardo Neves Bernardo.

Next, you can find an interview with Micha Mazaheri, CEO of Paw Inc. We have spoken about their tool and the power of start-ups.

Lastly we have 3 pieces for our columns. First we have an article from Damian Czernous' column "Model View Whatever - MVP by Mike Potel." We then have a TrueNAS Street column about "Solving Storage challenges with Root on ZFS in FreeNAS and TrueNAS." Finally, this issue closes out with Rob Somerville's column about his point of view on the FBI vs Apple case.

Enjoy your Easter Holidays, St. Patricks Day, Norooz, Holi and Magha Puja Day! And we hope you had a very spooky Nyepi ;)

Marta & BSD Team

MAGAZINE **BSD**

Editor in Chief:

Marta Ziemianowicz

marta.ziemianowicz@software.com.pl

Contributing:

David Carlier, Siju Oommen, Damian Czernous, Michael Boelem, Valerie Heatley, Mark VonFange, Roger Pau Monne and Rob Somerville.

Top Betatesters & Proofreaders:

Annie Zhang, Denise Ebery, Eric Geissinger, Luca Ferrari, Imad Soltani, Olaoluwa Omokanwaye, Radjis Mahangoe, Mani Kanth, Ben Milman, Mark VonFange and David Carlier

Special Thanks:

Annie Zhang

Denise Ebery

DTP:

Marta Ziemianowicz

Senior Consultant/Publisher:

Paweł Marciniak

pawel@software.com.pl

CEO:

Joanna Kretowicz

joanna.kretowicz@software.com.pl

Publisher:

Hakin9 Media SK 02-676 Warsaw, Poland Postepu 17D Poland worldwide publishing editors@bsdmag.org www.bsdmag.org

Hakin9 Media SK is looking for partners from all over the world. If you are interested in cooperation with us, please contact us via e-mail: editors@bsdmag.org.

All trademarks presented in the magazine were used only for informative purposes. All rights to trademarks presented in the magazine are reserved by the companies which own them.

CONTENTS

News

BSD World Monthly News 6

by Marta Ziemianowicz

This column presents the latest news coverage of breaking news events, products releases and trending topics of the BSD new stories.

Swift

Introduction to Swift, Open Source Language for iOS 14

by David Carlier

With this article, we'll describe how to write an Apple iPhone Operating System's application with some open source tools. Swift is now the most recent programming language and happily was open sourced since the end of 2015. Swift not only runs on Apple devices but also on Linux and a FreeBSD version is a work in progress. Experience with another language with similar features, like Ruby, Groovy, Rust, etc., can be helpful.

Open Source Might Be the New Business Era! 27

by Georgios Pessios

Back in June of 2014, during WWDC (Worldwide Developers Conference), Apple announced its new programming language; Swift! The creation of the language started in the summer of 2010 by Chris Lattner. Such a move was a surprise for the software coder community but Lattner kept his newborn masterpiece a secret from everyone, even from his top executives for more than a year and half.

Memory Management in Swift 30

by Arash Z. Jahangiri

Managing how long an object is going to stay alive in memory, before its memory is disposed, is called "Life Cycle Management". In some languages like Java this is done automatically. That is, GC (Garbage Collector) is responsible for releasing the objects that are no longer used from memory, in order to prevent a memory leak. As a result, the memory is

disposed and is ready to be used by other objects if needed. However, in Swift, ARC (Automatic Reference Counting) does this task by keeping track of memory, and releasing the memory as soon as it finds an object with no references to it.

FreeBSD

Why FreeBSD? 44

by Hamza Sheikh

Before going into why FreeBSD is now my preferred OS for learning UNIX, let us review why I used Linux for a long time.

Samhain

Samhain - Free, Open Source File Integrity Monitoring / HIDS 48

by Leonardo Neves Bernardo

One of the defining features of UNIX is 'Everything is a file', and one of the defining features of its administration is 'You cannot manage what you do not measure'. So, if we synthesize these phrases, we can conclude that 'It is necessary to measure our files to be a UNIX administrator'. Well, let us exchange the term measure for monitor, because in an operating system we can 'measure' a lot of features related to files. For example, we can monitor content, owner, and MAC times. The conclusion is 'We can be good UNIX administrators only if we can monitor files'.

In this article we will learn how to use the Samhain software to monitor activities in the UNIX operating system, above all, to monitor file modifications.

Interview

Interview with Micha Mazaheri, CEO of Paw 63

by Marta Ziemianowicz, Marta Strzelec & Marta Si-enicka

CONTENTS

GUI

Model View Whatever - MVP by Mike Potel 66

by Damin Czernous

In the 1990s, software engineers attempt to unite two dominating UI designs: Forms and Controls, and MVC with the Application Model (AMVC). The result is that the Model View Presenter (MVP) structure emerges. It is difficult to determine the MVP inventor. It seems to be a collective work occasionally summarized by specific engineers. In that design cloud, two mainstreams start clashing. One sympathizes with Mike's Potel's way of thinking. The other one promotes Andy Bower and Blair McGlasha's point of view. Generally however, MVP is referenced via the Potel paper MVP, "Model-View-Presenter The Taligent Programming Model for C++ and Java".

TrueNAS Street

Solving Storage Challenges with Root on ZFS in FreeNAS and TrueNAS 73

by Mark VonFange

The ZFS file system provides data integrity features for storage drives using its Copy On Write (CoW) technology and improved RAID, but these features have been limited to storage drives previously. If you have a drive failure, utilizing RAID or mirroring will protect your volumes, but what happens if your boot drive fails?

Rob's Column 75

by Rob Somerville

The FBI and Apple are engaged in very public spat concerning encryption, data privacy and intellectual property. Who should we be more afraid of – government, business or terrorists?

BSD Certification

The BSD Certification Group Inc. (BSDCG) is a non-profit organization committed to creating and maintaining a global certification standard for system administration on BSD based operating systems.

? WHAT CERTIFICATIONS ARE AVAILABLE?

BSDA: Entry-level certification suited for candidates with a general Unix background and at least six months of experience with BSD systems.

BDSP: Advanced certification for senior system administrators with at least three years of experience on BSD systems. Successful BDSP candidates are able to demonstrate strong to expert skills in BSD Unix system administration.

✓ WHERE CAN I GET CERTIFIED?

We're pleased to announce that after 7 months of negotiations and the work required to make the exam available in a computer based format, that the BSDA exam is now available at several hundred testing centers around the world. Paper based BSDA exams cost \$75 USD. Computer based BSDA exams cost \$150 USD. The price of the BDSP exams are yet to be determined.

Payments are made through our registration website:
<https://register.bsdcertification.org/register/payment>

i WHERE CAN I GET MORE INFORMATION?

More information and links to our mailing lists, LinkedIn groups, and Facebook group are available at our website:
<http://www.bsdcertification.org>

Registration for upcoming exam events is available at our registration website:
<https://register.bsdcertification.org/register/get-a-bsdcg-id>

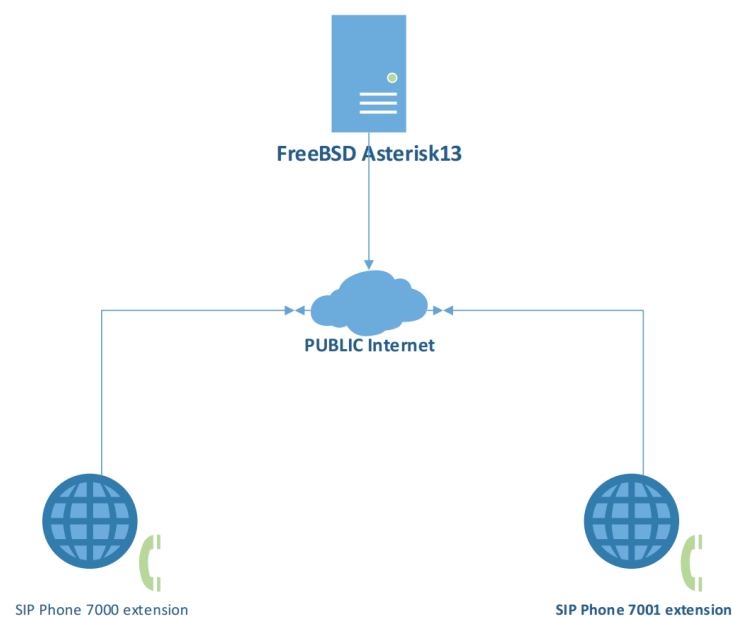
FreeBSD Asterisk test between 2 extensions

Asterisk — Open source PBX system. Developed by Mark Spencer. Working on Linux, FreeBSD, OpenBSD and Solaris operation systems. The project name created from “*”. You can read more about asterisk from <http://asterisk.ru/> and <http://www.asterisk.org/> official page.

Asterisk has all the features that are in any hardware PBX and supports many VoIP protocols.

Features:

- Voice Mail
- Conference rooms
- IVR menu
- Call handling center (by using different algorithms, define call queueing and distribution to subscribers)
- Call Detail Record



<http://www.unixmen.com/freebsd-asterisk-test-between-2-extensions>

Microsoft's Red Hat Ansible love-in gives birth to DevNetCLOps

Only kidding. But it is a world domination plan, y'know

As we know, Hell froze over a while ago: Red Hat and Microsoft are now friends. The latest chapter saw Red Hat point its newly acquired Ansible IT automation technologies towards networks, clouds and Windows environments, because who wouldn't want a slice of the Azure pie, now that Microsoft loves Linux?

Red Hat's penchant for Azure is part of a DevOps world domination plan where the open-source champion's first gambit is to extend Ansible towards network infrastructure devices. But let's go back to the very start of the story first.

Here's the play

The play here is: bring DevOps to cloud-centric networking so that we can orchestrate entire application infrastructures, including network devices, with one single automation tool.

How does this work? Well, Ansible (sorry, Red Hat's Ansible) will provide native agentless support for automating heterogeneous network infrastructure devices using the same human and machine-readable automation language that Ansible provides to IT teams.

Using less technical language, suddenly this starts to sound like intelligent automation control without the need to custom code internal system agents (hence the agentless part) not just for software application code, but also for network devices from the likes of Arista, Cisco and Juniper.

Ansible's automation language is said to be "easier to learn" than the custom scripting required to build a software agent (hence, the agentless part, again). So taking this power upwards into the network could help serve internal system needs, such as initiating, monitoring and terminating other software applications or indeed other agents and, crucially, now, other network components.

http://www.theregister.co.uk/2016/02/25/red_hat_ansible_azure_networks_devops/

Intel shows budget Android phone powering big-screen Linux

Move over Continuum: No customers yet, but Intel's ready and waiting

MWC16 Intel is showing what it calls "Big Screen Experience" at Mobile World Congress, an Android smartphone which runs a full Linux desktop when plugged into an external display. The concept is broadly similar to Microsoft's Continuum for Windows 10 Mobile, but whereas Continuum devices are towards the high end, Intel's project is aimed, it says, at budget smartphones and emerging markets.



On display in Barcelona is a prototype SoFIA (Smart or Feature Phone with Intel Architecture) smartphone with an Atom x3 processor, 2GB RAM and 16GB storage, and modified to support an external display. Attach keyboard, mouse and display, and it becomes desktop Linux, with an option to display the Android screen in a window on the large display.

"Android is based on a Linux kernel, so we're running one kernel, we have an Android stack and a Linux

stack, and we're sharing the same context, so the file system is identical. The phone stays fully functional," Intel's Nir Metzger, Path Finding Group Manager, told the Reg.

http://www.theregister.co.uk/2016/02/23/move_over_continuum_intel_shows_android_smartphone_powering_bigscreen_linux/

iOS app that smuggled pirated software into China is booted out of Apple's walled garden

Bootlegger tool posed as English language guide

A dodgy application that evaded Apple's hardline code reviewers and made it into Cupertino's official App Store has been turfed out.

The program – which featured a hidden smugglers' cove of software – was ejected after it was fingered by third-party security researchers.

The team at Palo Alto Networks explained over the weekend that although the Happy Daily English app posed as an English studying tool to users outside China, it actually offered a secret store of pirated and cracked iOS apps and games to users within China.

Apple's official iOS App Store is well known for its strict code review process. This mandatory policy for app acceptance into Apple's walled garden has become an important mechanism in protecting the privacy and security of iOS users. Circumventing this policy therefore shakes the foundation of the whole App Store ecosystem.

http://www.theregister.co.uk/2016/02/22/risky_apple_app_banhammer/

OpenBSD website operators urged to fix mind-alteringly bad bug

COMIC SANS has infiltrated the HTTP daemon. Thank Cthulu a patch has been delivered.

Someone has offered* OpenBSD's maintainers an important peace-of-mind patch for the operating system's HTTP daemon.

It's not a security exposure they've fixed, but something vastly worse: the daemon defaulted to Comic Sans for its 404 "page not found" messages.

You think that's not bad? Here's how Peter Krantz, who fixed the bug, put it:

"For some reason the httpd status pages (e.g. 404) use the Comic Sans typeface. This patch removes comic sans and sets the typeface to the default sans-serif typeface of the client.

This lowers the number of people contacting website maintainers with typeface complaints bordering on harassment" (emphasis added).

Krantz is being thorough in his defence of sysadmins' peace of mind. The fix removes three fonts from the list, if Vulture South has read it correctly: Comic Sans MS, Chalkboard SE, and Comic Neue.

Clearly, such an egregious bug could not have happened by accident.

As a later poster to the list noted, the choice of Comic Sans was an attempt to solicit donations: the individual who made the choice wanted to "annoy hipsters" into donating to the project.

The "weaponised Comic Sans" joke lasted 14 years. ®

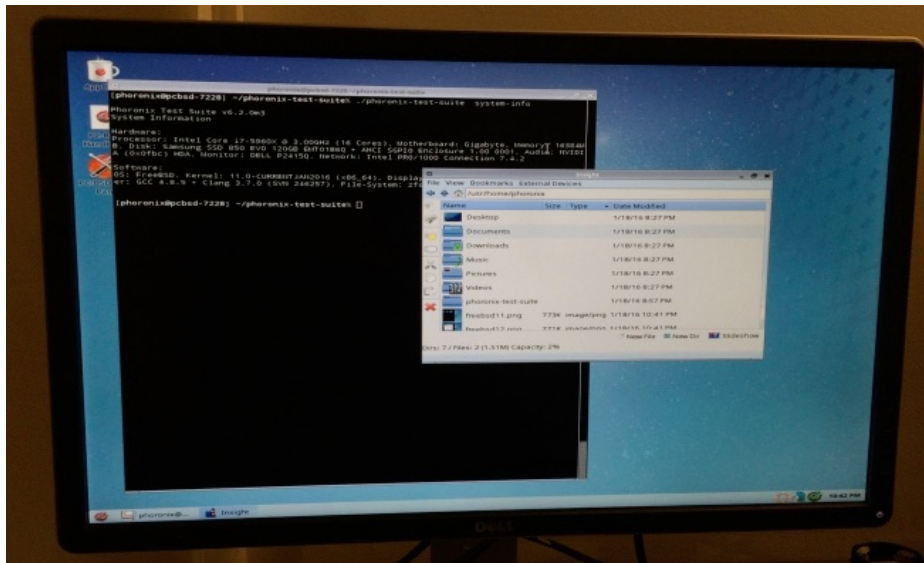
*Footnote: The patch isn't official, and by mis-attributing it, even humorously, to OpenBSD maintainers, Vulture South has apparently committed a breach of Ethics in OpenBSD journalism.

http://www.theregister.co.uk/2016/02/21/openbsd_website_operator_patch_now_for_the_sake_of_your_sanity/

PC-BSD / FreeBSD 11.0-CURRENT Performance

From Michael Larabel:

For those that don't know, FreeBSD boasts a Linux binary compatibility initiative. Five years ago, I did some Linux gaming tests on FreeBSD within FreeBSD: A Faster Platform For Linux Gaming Than Linux?. I wanted to do some modern tests atop the latest FreeBSD/PC-BSD code and the latest NVIDIA driver.



So I decided to go for this month's PC-BSD 11.0-CURRENT release to get the bleeding-edge state of the FreeBSD performance and for best Linux binary compatibility. While PC-BSD ships with the support enabled by default, and I did install all of the relevant CentOS-derived packages from Ports, I couldn't get any of my usual OpenGL Linux game / tech demo benchmarks (32-bit, since the 64-bit Linux binary support on FreeBSD is incomplete) running under 11.0-CURRENT. All my attempts were foiled by segmentation faults.

Five years ago, I got the FreeBSD Linux binary compatibility support working real good, as shown in those earlier results, but whatever the case, I couldn't get it working well on FreeBSD 11.0-CURRENT. After a few hours, I had to throw in the towel and focus on other work. However, in not to spoil having a clean FreeBSD 11.0-CURRENT installation around, I did some quick benchmarks of this latest release.

PC-BSD 11.0-CURRENT with this month's image was using the 11.0-CURRENT kernel (obviously), GCC 4.8.5 and Clang 3.7 were setup as the compiler stack, and ZFS continues to be the default PC-BSD file-system. For putting the PC-BSD 11.0-CURRENT results into some perspective, I then installed Fedora 23 x86_64 on this same system: comprised of an Intel Core i7 5960X, 16GB of RAM, 120GB Samsung 850 EVO SSD, and NVIDIA GeForce GTX 750 Ti graphics.

If you want to see these PC-BSD 11.0-CURRENT vs. Fedora 23 Linux benchmarks from this Core i7 Haswell system, see this OpenBenchmarking.org result file for the few results to share today. As FreeBSD/PC-BSD 11.0 is nearing its official release, plenty more thorough benchmarks will obviously come.

https://www.phoronix.com/scan.php?page=news_item&px=FreeBSD-11.0-Jan-CURRENT

Lumina Desktop Getting Ready for FreeBSD 11.0

It appears the sun is rising on Lumina.

Ken Moore, the lead developer for the BSD-based Lumina Desktop Environment, announced that another step towards the release of a full-fledged desktop environment for BSD variants (and Linux distros, for that matter) has been achieved with the release of version 0.8.8 yesterday.

For those of you keeping score at home, the Lumina Desktop Environment — let's just call it Lumina for short — is a lightweight, XDG-compliant, BSD-licensed desktop environment focusing on getting work done while minimizing system overhead.



Specifically designed for PC-BSD and FreeBSD, it has also been ported to many other BSD variants and Linux distros. Lumina is based on the Qt graphical toolkit and the Fluxbox window manager, and uses a small number of X utilities for various tasks.

Moore said in his announcement that he, along with his team, are targeting the release of Lumina version 1.0 to coincide with the release of FreeBSD 11.0 in July, “but that

schedule is also subject to change a bit as needed down the road.”

“This release primarily contains bug fixes, interface tweaks, and some small expansions of current functionality,” Moore writes. “In addition to this, we also now have support for NetBSD systems out of the box.”

A list of developments – including feature updates, translation improvements and bug fixes – can be found on the announcement page. For further information on the desktop environment, you can visit this page.

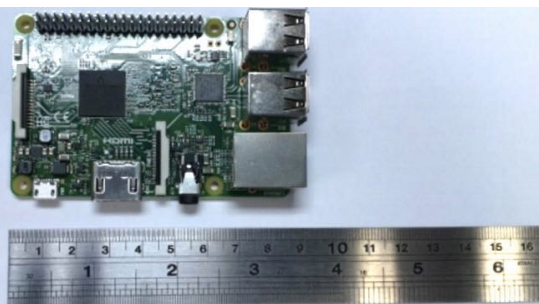
Many of you know me as dyed-in-the-wool Xfce guy, but I’m no stranger to window managers like Fluxbox (and Openbox for that matter, which CrunchBang used back in the day). You can be sure I’m looking forward to giving Lumina a shot once it’s ready for prime time, and I’ll give you the details once I do.

One more thing: It happened again. Apparently BSD might be too scary for the general public – at least in the cafes I frequent. My early-morning MO at one cafe which, sadly, isn’t Firefly Coffee House (it’s not open this early, Caitlin – I’m sorry!) entails my setting up my ThinkPad, pushing the on button, and then going to the counter to order. Earlier this week – Monday to be exact – a guy yelled at me from the back of the cafe, “Hey, man, you’re computer’s freaking out.” I walked back to the ThinkPad to see PC-BSD going through its normal start-up paces, and I explained to the concerned fellow that this is normal for the operating system.

“It’s telling me exactly what it’s doing,” I explained, though it did start me on a search for a “quiet” boot mode, if one exists. Cue the helpful BSD folks to offer multiple suggestions, which is one of the great things I am finding about this community.

<http://fossforce.com/2016/02/lumina-desktop-getting-ready-freebsd-11-0/>

Raspberry Pi 3 Released With Built-in Wi-Fi, Bluetooth, and 50 Percent Faster Processor



Continuing its tradition of releasing new flagship products in February, the Raspberry Pi Foundation has launched Raspberry Pi 3 with many improvements. The new single board PC comes with onboard Wi-Fi, Bluetooth 4.1, and a 50 percent performance improvement.

The Raspberry Pi Foundation has just unveiled a brand new Raspberry Pi with many improvements. The launch of tiny

Raspberry Pi 3 also celebrated the fourth anniversary of the Pi family.

The new single board computer comes with Wi-Fi and Bluetooth, and it still costs \$35. “Wi-Fi and Bluetooth are a thing people have been asking for for a long time,” says Eben Upton, CEO of Raspberry Pi Trading, the Raspberry Pi Foundation’s commercial arm.

With a low price tag of \$35, Raspberry Pi has been able to revolutionize the programming and electronics learning in schools. Raspberry Pi 3 will allow users to ditch the external Wi-Fi USB antenna and Ethernet cables, making this affordable PC more usable. The onboard Bluetooth will allow the makers and researchers to collect data using various sensors, leaving behind the need to deal with multiple wires.

Raspberry Pi has also gained a lot in terms of speed. A new 1.2 GHz 64-bit processor gives it a 50 percent boost in terms of speed and performance.

Key improvements and features – Raspberry Pi 3:

- A 1.2GHz 64-bit quad-core ARM Cortex-A53 CPU (~10x the performance of Raspberry Pi 1)
- Integrated 802.11n wireless LAN and Bluetooth 4.1
- Complete compatibility with Raspberry Pi 1 and 2
- Raspberry Pi 3 ships with 32-bit Noobs OS
- Same 1GB of RAM and 400 MHz VideoCore IV graphics

“I’m really quite hopeful that this time we might come across that line that we’ve been trying to cross for a long time,” Upton says. “That we’ve made a thing where you can really say, ‘Yes, this is a PC.’”

<http://fossbytes.com/raspberry-pi-released-with-wi-fi-bluetooth/>

Great Specials

On FreeBSD® & PC-BSD® Merchandise

Give us a call & ask about our
SOFTWARE BUNDLES

1.925.240.6652

\$39.95

FreeBSD 9.1 Jewel Case CD Set
or FreeBSD 9.1 DVD

\$29.95

PC-BSD 9.1 DVD

\$49.95

The PC-BSD 9.0 Users Handbook
PC-BSD 9.1 DVD

\$99.95

The FreeBSD CD or DVD Bundle

Inside each CD/DVD Bundle, you'll find:
FreeBSD Handbook, 3rd Edition
Users Guide FreeBSD Handbook, 3rd Edition, Admin Guide
FreeBSD 9.1 CD or DVD set
FreeBSD Toolkit DVD



Stylish Dress Attire
Look Your Professional Best



Comfy Apparel
Stay Warm in Zip Ups & Pullovers

T-Shirts
Lots of Styles to Choose From

FreeBSD 9.1 Jewel Case CD/DVD \$39.95

CD Set Contains:

- Disc 1** Installation Boot LiveCD (i386)
- Disc 2** Essential Packages Xorg (i386)
- Disc 3** Essential Packages, GNOME2 (i386)
- Disc 4** Essential Packages (i386)

FreeBSD 9.0 CD \$39.95

FreeBSD 9.0 DVD \$39.95

FreeBSD Subscriptions

Save time and \$\$\$ by subscribing to regular updates of FreeBSD

FreeBSD Subscription, start with CD 9.1 \$29.95

FreeBSD Subscription, start with DVD 9.1 \$29.95

FreeBSD Subscription, start with CD 9.0 \$29.95

FreeBSD Subscription, start with DVD 9.0 \$29.95

PC-BSD 9.1 DVD (Isotope Edition)

PC-BSD 9.1 DVD \$29.95

PC-BSD Subscription \$19.95

The FreeBSD Handbook

The FreeBSD Handbook, Volume 1 (User Guide) \$39.95

The FreeBSD Handbook, Volume 2 (Admin Guide) \$39.95

The FreeBSD Handbook Specials

The FreeBSD Handbook, Volume 2 (Both Volumes) \$59.95

The FreeBSD Handbook, Both Volumes & FreeBSD 9.1 \$79.95

PC-BSD 9.0 Users Handbook \$24.95

BSD Magazine \$11.99

The FreeBSD Toolkit DVD \$39.95

FreeBSD Mousepad \$10.00

FreeBSD & PCBSD Caps \$20.00

BSD Daemon Horns \$2.00



Bundle Specials!
Save \$\$\$

Just Plain Fun
Mousepads & Novelty Horns



BSD Magazine
Available Monthly



For even MORE items
visit our website today!

www.FreeBSDMall.com

Introduction to Swift, Open Source Language for iOS

by David Carlier

With this article, we'll describe how to write an Apple iPhone Operating System's application with some open source tools. Swift is now the most recent programming language and happily was open sourced since the end of 2015. Swift is not only runs on Apple devices but also on Linux and a FreeBSD version is a work in progress. Experience with another language with similar features, like Ruby, Groovy, Rust, etc., can be helpful.

1. Swift characteristics

Swift gathers the performance of a compiled programming language while insuring to protect better against the common issues we find in the lower level languages; null pointers, memory leakage, uninitialized variables and so forth. Of course, closures, generics (in a similar manner as Java), and exceptions handling you can find in modern languages are present but there is a unique feature which protects against null pointer, as said earlier. It is called optional chaining. The optional chaining is simply an operator to place to an object which can be possibly null (nil in Swift terminology), hence avoiding a null exception.

```
class OtherClass {  
  
    var name = "John Doe"  
  
    var age = 25  
  
    ...  
  
}
```



```
class MyClass {
    var member: OtherClass?

    ...

    func display() {
        ...
        // If member is not initialized, with ? We avoid a null
        pointer exception

        // Conditionally the following block won't be executed in this case
        if let name = member?.name {
            print "I am " + member.name
        } else {
```

We just saw two different ways to declare a variable, let and var keywords. Let is for assigning constants (you do not need to specify the type) whereas var is for variables (in our case, member is a type of “OtherClass”).

```
protocol MyProtocol {

    // We can choose to allow a variable
    to have a getter and/or a setter

    var name : String { get }

    var age : Int { get }

    var available : Bool { get
    set }

    func calculateSalary(var
    base: Double, var prime: Double)
    -> Double

}
```

Another one is called Protocol, an object can be extended without inheritance. For those familiar with Java's interfaces, the concept is similar; we define a contract which a class (or a struct) must implement ; variables and functions.

```
class Person : MyProtocol {

    var name : String

    var age : Int

    var available : Bool

    func calculateSalary(var base: Double, var prime: Double) -> Double {

        return base + prime

    }

}
```

Struct and classes are different than their C++ counterparts; for example, a struct does not support inheritance and is passed by value, whereas a class is passed by reference to a function.

We mentioned the closures earlier; there are numerous forms, as we can see below:

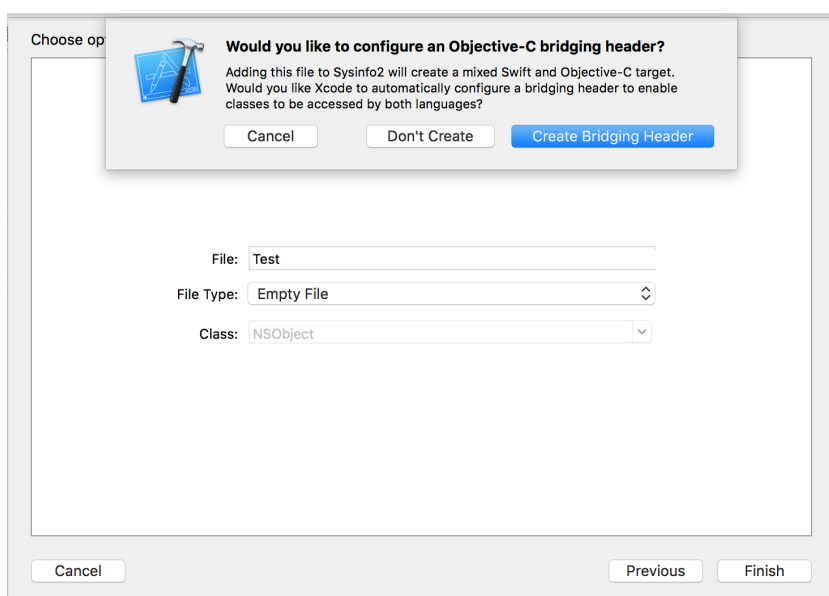
```
let printme = ({var name: String} print("My name is " + name))

printme("John Doe")

let a = ["Apple", "Strawberry", "Orange"]

// sort takes a comparator's function with this signature. A closure can perfectly fit this role

a.sort({var a: String, var b: String} -> in return a > s)
```



Objective-C code can be used inside Swift. Indeed, in order to import an existing Objective-C library, we just need the “bridge header” mechanism. This created file will expose all the public Objective-C to our Swift application.

Figure 1. Objective-C bridging header configuration.

Here we have our Objective-C code:



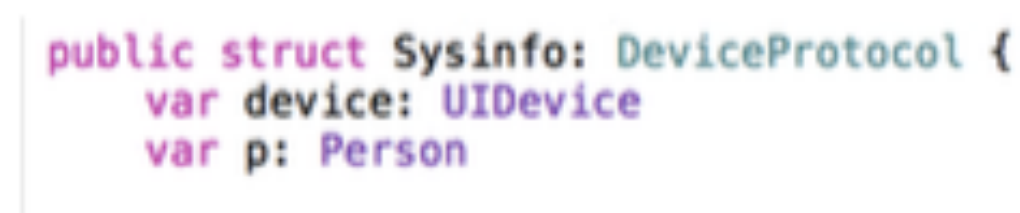
```

1 //
2 // Test.m
3 // Sysinfo2
4 //
5 // Created by Dclink on 24/02/2016.
6 // Copyright © 2016 BSDMag. All rights reserved.
7 //
8
9 #import <Foundation/Foundation.h>
10
11
12 @interface Person : NSObject
13 {
14     StringPtr name;
15     int age;
16 }
17 @end
18
19 @implementation Person
20
21 -(id)init
22 {

```

Figure 2. Objective-C code.

And in the Swift Code, we can create a Person instance without importing anything:



```

public struct Sysinfo: DeviceProtocol {
    var device: UIDevice
    var p: Person

```

Figure 3. Swift Code Person instance.

2. Getting Swift

a. Under Mac OS X

There are two ways to get Swift.

a/ Via Xcode

- Xcode 7.0 and above is shipped with a Swift compiler release so it is possible to create a whole project with this language.

b/ Via the source code

- It is also possible to compile Swift from the source, under OS X it is pretty handy.
- In order to build Swift, you need Xcode development tools, Git and Cmake.
- Now, we can clone this Git repository <https://github.com/apple/swift>

```
cd swift/  
  
Apple:swift$ ./utils/update-checkout --clone  
  
--- Cloning 'cmark' ---  
  
Cloning into 'cmark'...  
  
remote: Counting objects: 8300, done.  
  
remote: Total 8300 (delta 0), reused 0 (delta 0), pack-reused 8300  
  
Receiving objects: 100% (8300/8300), 2.78 MiB | 1.74 MiB/s, done.  
  
Resolving deltas: 100% (5753/5753), done.  
  
Checking connectivity... done.  
  
--- Cloning 'lldb' ---  
  
Cloning into 'lldb'...
```

After a couple of minutes, all dependencies are cloned as well, and we can start to build. For Xcode, you can simply type:

```
Apple:swift$ utils/build-script -x  
  
Building the standard library for: swift-stdlib-macosx-x86_64
```



```
mark: using standard linker

++ cmake_config_opt cmark

++ product=cmark

++ [[ Xcode == \X\c\o\d\e ]]

++ echo '--target ZERO_CHECK '

++ case ${product} in

++ echo '--config Debug'

+ /usr/local/bin/cmake --build /Users/Apple/build/Xcode-DebugAssert/
cmark-macosx-x86_64 --target ZERO_CHECK --config Debug

=== BUILD AGGREGATE TARGET ZERO_CHECK OF PROJECT cmark WITH CONFIGURA-
TION Debug ===
```

Check dependencies:

```
PhaseScriptExecution CMake\ Rules
/Users/Apple/build/Xcode-DebugAssert/cmark-macosx-x86_64/cmark.build/
Debug/ZERO_CHECK.build/Script-7E3497D36F7744189435A439.sh

    cd /Users/Apple/cmark

    /bin/sh -c
/Users/Apple/build/Xcode-DebugAssert/cmark-macosx-x86_64/cmark.build/
Debug/ZERO_CHECK.build/Script-7E3497D36F7744189435A439.sh

echo ""

make -f
/Users/Apple/build/Xcode-DebugAssert/cmark-macosx-x86_64/CMakeScripts
/ReRunCMake.make

make[1]:
`/Users/Apple/build/Xcode-DebugAssert/cmark-macosx-x86_64/CMakeFiles/
cmake.check_cache' is up to date.

** BUILD SUCCEEDED **
```

b. Under Linux

Here we will be focusing on Ubuntu which has Swift snapshot versions which eases the installation.

First, we need these dependencies:

- git
- cmake
- ninja-build
- clang
- uuid-dev
- libicu-dev
- icu-devtools
- libbsd-dev
- libedit-dev
- libxml2-dev
- libsqlite3-dev
- s w i g

Once all those dependencies are installed, it is now possible to download a snapshot from this URL:

<https://swift.org/download/#latest-development-snapshots>

Depending to your Ubuntu version, there are two links available. Once downloaded and extracted, it might be more handy to add the path of the Swift executable to your path:

```
export PATH=$PATH:<path to the extracted Swift archive>/usr/bin
```


c. FreeBSD

For FreeBSD, we would need to install these dependencies :

- git
- cmake
- ninja
- clang36
- libc++
- icu
- libxml2
- sqlite3
- swig
- python27
- ncurses
- pkgconf

After that, to clone this git repository locally:

git clone <https://github.com/landonf/swift-freebsd>

Then you need to have clang36 and clang++36 environment variables set explicitly, as below:

```
export HOST_CC=clang36  
export HOST_CXX=clang++36
```

Then clone these Swift subprojects as follows:

```
git clone https://github.com/apple/swift.git swift
```

```
git clone https://github.com/apple/swift-llvm.git llvm
```

```
git clone https://github.com/apple/swift-clang.git clang
```

```
git clone https://github.com/apple/swift-lldb.git lldb
```

```
git clone https://github.com/apple/swift-cmark.git cmark
```

```
git clone https://github.com/apple/swift-llbuild.git llbuild
```

```
git clone https://github.com/apple/swift-package-manager.git swiftpm
```

```
git clone https://github.com/apple/swift-corelibs-xctest.git
```

```
git clone https://github.com/apple/swift-corelibs-foundation.git
```

```
git clone https://github.com/martine/ninja.git
```

Then `utils/build-script -t`

3. Writing a first application

As a first application, let's write a very basic iOS application which displays some of the device information, like the current model and the OS version. Let's open a new project for this purpose.

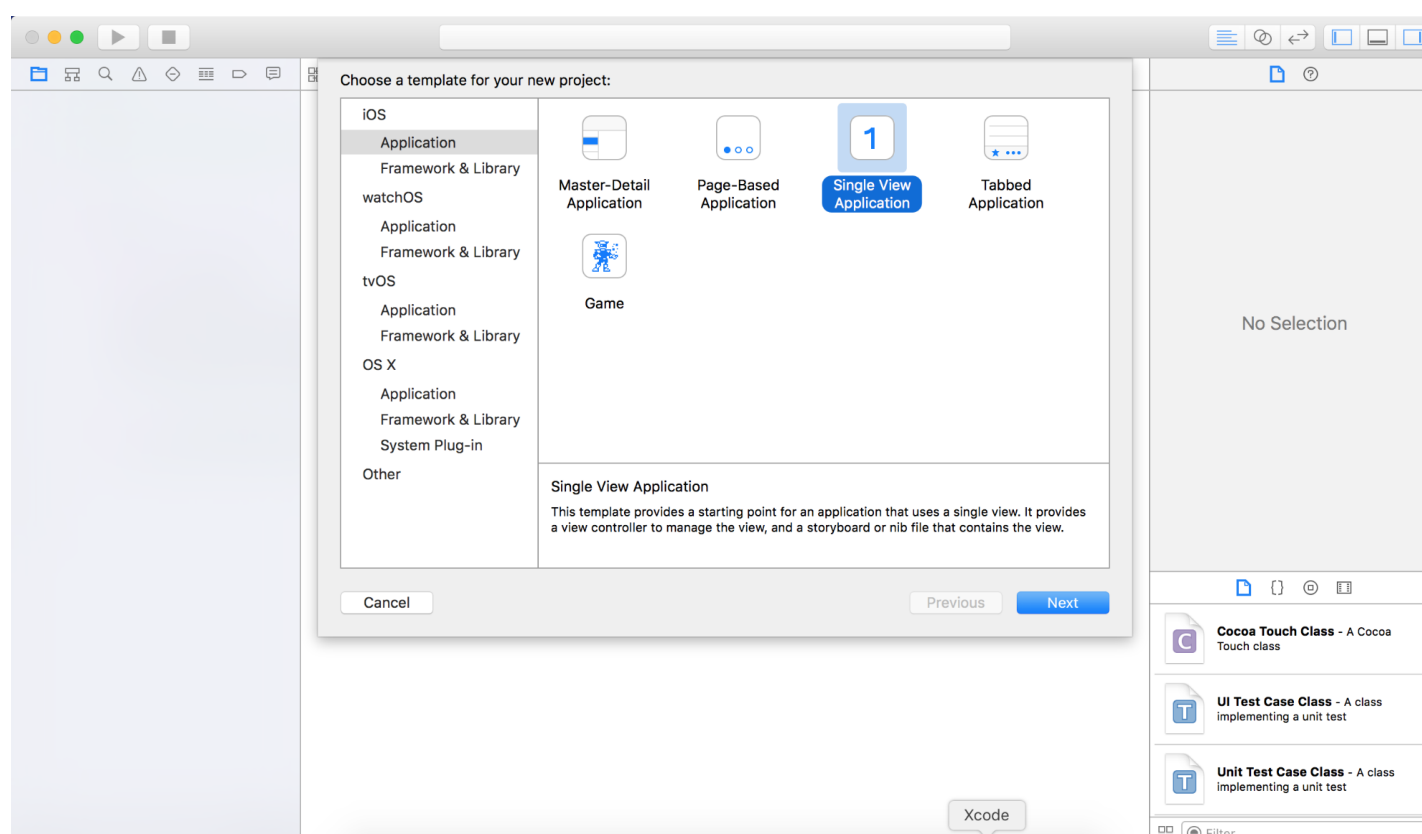


Figure 4. Single View Application.

First, we create a protocol with few functions.

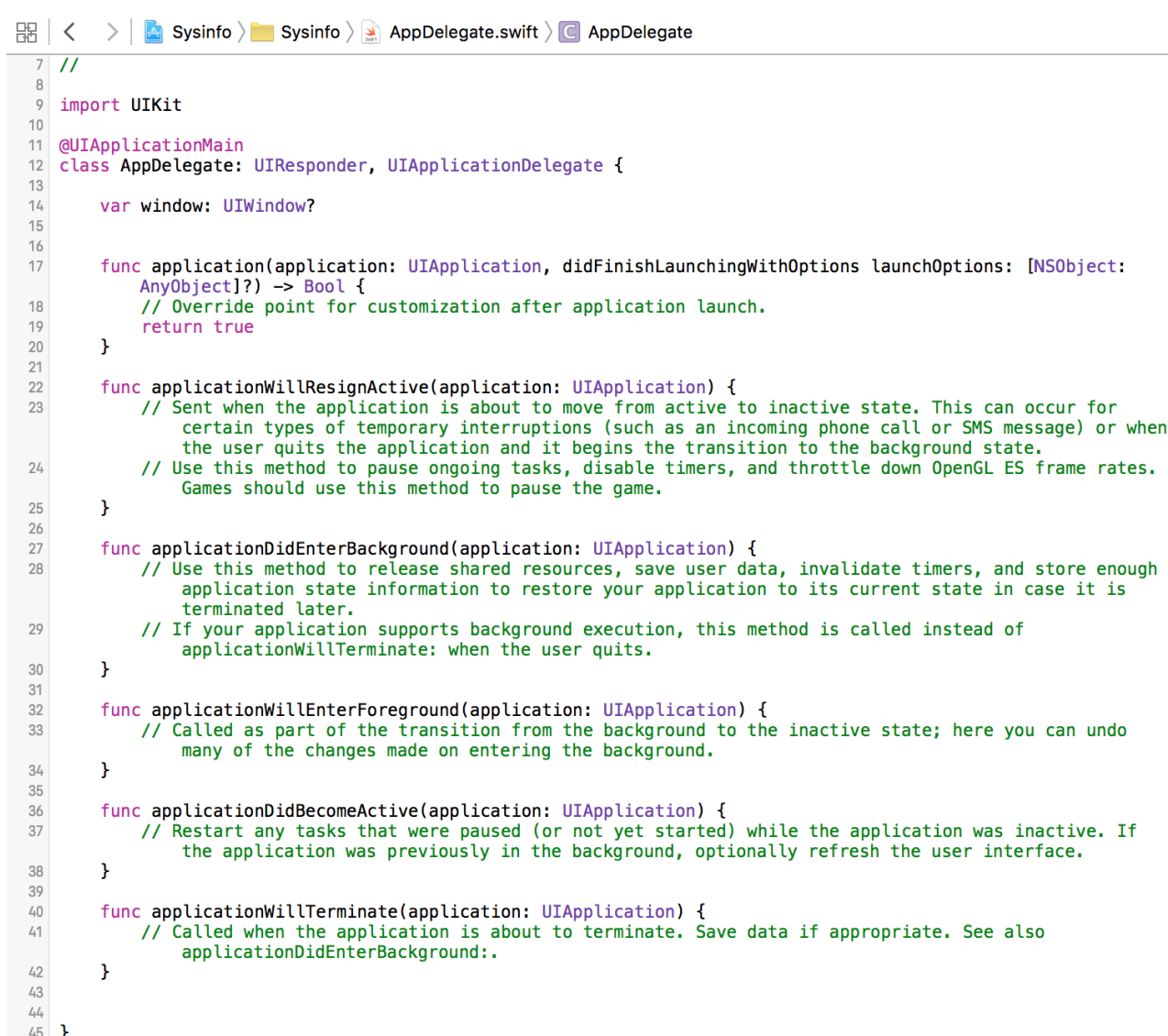


```

1 //
2 // DeviceProtocol.swift
3 // Sysinfo
4 //
5 // Created by Dclink on 22/02/2016.
6 // Copyright © 2016 BSDMag. All rights reserved.
7 //
8
9 import Foundation
10 import UIKit
11
12 protocol DeviceProtocol {
13     var device : UIDevice { get }
14     func model() -> String
15     func osVersion() -> String
16     func UID() -> String
17 }
    
```

Figure 5. Creating a new protocol.

Then let's create a plain Swift struct which implements the protocol and holds the device's information.



```

7 //
8
9 import UIKit
10
11 @UIApplicationMain
12 class AppDelegate: UIResponder, UIApplicationDelegate {
13
14     var window: UIWindow?
15
16
17     func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
18         // Override point for customization after application launch.
19         return true
20     }
21
22     func applicationWillResignActive(application: UIApplication) {
23         // Sent when the application is about to move from active to inactive state. This can occur for
24         // certain types of temporary interruptions (such as an incoming phone call or SMS message) or when
25         // the user quits the application and it begins the transition to the background state.
26         // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates.
27         // Games should use this method to pause the game.
28     }
29
30     func applicationDidEnterBackground(application: UIApplication) {
31         // Use this method to release shared resources, save user data, invalidate timers, and store enough
32         // application state information to restore your application to its current state in case it is
33         // terminated later.
34         // If your application supports background execution, this method is called instead of
35         // applicationWillTerminate: when the user quits.
36     }
37
38     func applicationWillEnterForeground(application: UIApplication) {
39         // Called as part of the transition from the background to the inactive state; here you can undo
40         // many of the changes made on entering the background.
41     }
42
43     func applicationDidBecomeActive(application: UIApplication) {
44         // Restart any tasks that were paused (or not yet started) while the application was inactive. If
45         // the application was previously in the background, optionally refresh the user interface.
46     }
47
48     func applicationWillTerminate(application: UIApplication) {
49         // Called when the application is about to terminate. Save data if appropriate. See also
50         // applicationDidEnterBackground:.
51     }
52 }
    
```

Figure 6. Creating a plain Swift.

Here is an autogenerated class, the main entry point of the application:

```

1 //
2 // Sysinfo.swift
3 // Sysinfo
4 //
5 // Created by Dclink on 22/02/2016.
6 // Copyright © 2016 BSDMag. All rights reserved.
7 //
8
9 import Foundation
10 import UIKit
11
12 public struct Sysinfo: DeviceProtocol {
13     var device: UIDevice
14
15     public init() { device = UIDevice() }
16     public func model() -> String { return device.model }
17     public func osVersion() -> String { return device.systemVersion }
18     public func UID() -> String { return String(device.identifierForVendor) }
19 }

```

Figure 7. Autogenerated class.

```

1 //
2 // ViewController.swift
3 // Sysinfo
4 //
5 // Created by Dclink on 30/01/2016.
6 // Copyright © 2016 BSDMag. All rights reserved.
7 //
8
9 import UIKit
10
11 class ViewController: UIViewController {
12     var sysinfo: Sysinfo?
13
14     override func viewDidLoad() {
15         super.viewDidLoad()
16         sysinfo = Sysinfo()
17         let label = UILabel(frame: CGRectMake(0, 0, 200, 21))
18         label.center = CGPointMake(160, 284)
19         label.textAlignment = NSTextAlignment.Center
20         label.text = (sysinfo?.model())! + " " + (sysinfo?.osVersion())!
21         self.view.addSubview(label)
22         // Do any additional setup after loading the view, typically from a nib.
23     }
24
25     override func didReceiveMemoryWarning() {
26         super.didReceiveMemoryWarning()
27         // Dispose of any resources that can be recreated.
28     }
29
30 }
31
32
33

```

Here the controller creates a text label, we just print the information.

Figure 8. Creating a text label.

Finally, we are able to compile the project and launch the iOS simulator, regardless of the UI type chosen. Here a Single-View layer:



Figure 9. A Single - View layer.

... Or a Page-Based application.

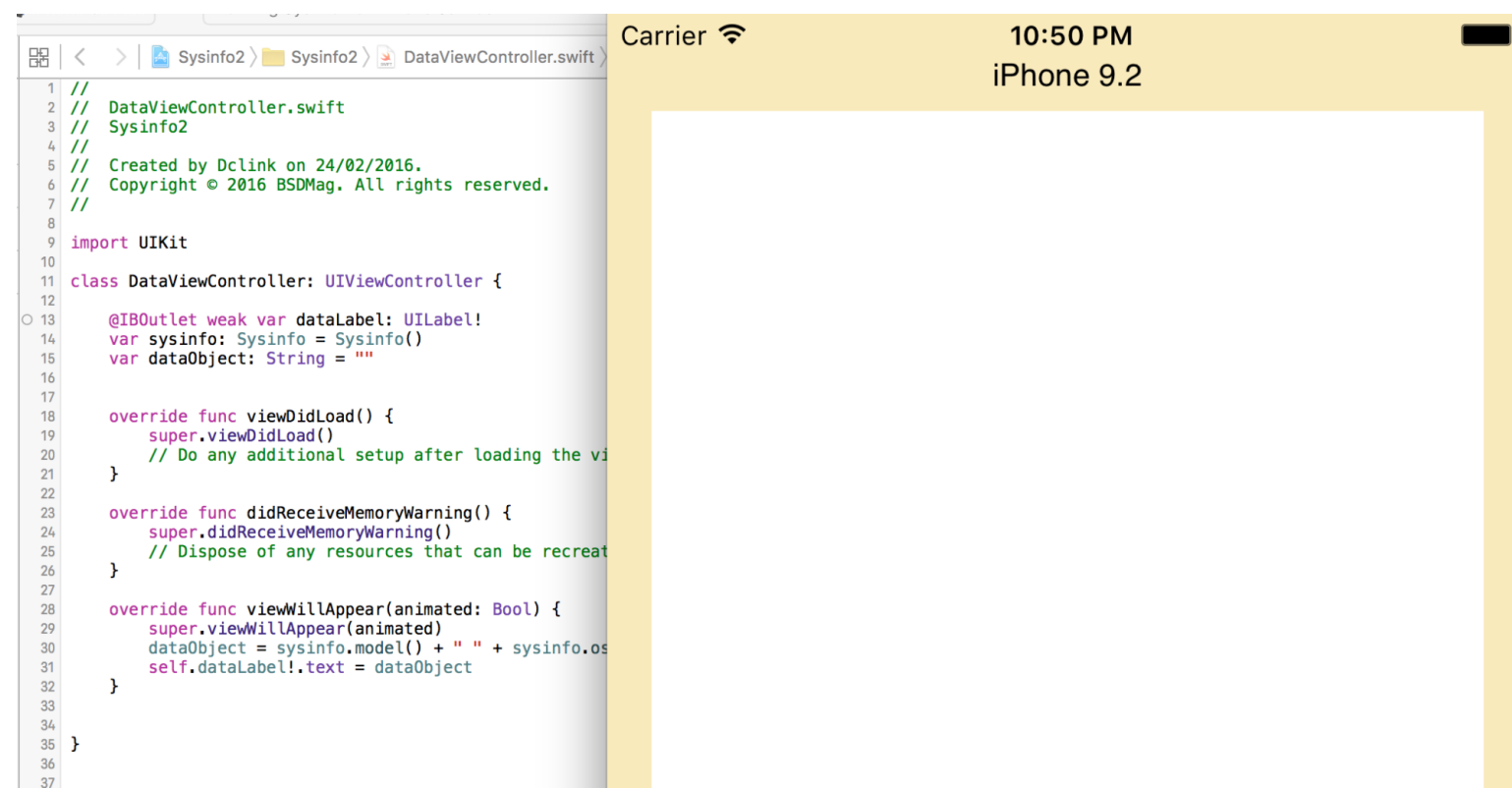


Figure 10. Page - Based application.

4. Going further

I hope this short introduction to Swift will give you the taste to go further in the Swift documentation, which you can find here <https://developer.apple.com/swift/resources/> Hopefully, the FreeBSD version will be improved and will match other versions, as well.



About the Author:

David Carlier is a developer since 2001, mainly C/C++, living and working in Ireland mainly since 2012. He contributes to some open source projects and uses in a daily basis various operating systems mainly BSD flavours.

Open Source Might Be the New Business Era!

by Georgios Pessios

Back in June of 2014, during WWDC (Worldwide Developers Conference), Apple announced its new programming language; Swift! The creation of the language started in the summer of 2010 by Chris Lattner. Such a move was a surprise for the software coder community but Lattner kept his newborn masterpiece a secret from everyone, even from his top executives for more than a year and half.

Swift reached a mass audience from the first moment. Though we are already talking about surprises, we didn't mention the best of all that was meant to happen about a year and a half later. On December 3rd of 2015, Apple made one step forward and open sourced Swift! That was a big "Hooray!!!" and "What???" at the same time for developers of "i-Devices" applications.

Traditionally, Apple was one of the most closed ecosystems in the world of technology. In a way, it still is and as written by others that was always one of the biggest competitive advantages. But it seems that something has started to change.

Before we go on, it's worth mentioning two terms at this point.

-The "Halo Effect": A cognitive bias in which an observer's overall impression of a person, company, brand, or product influences the observer's feelings and thoughts about that entity's character or properties. (Wikipedia)

-"Linus's Law": "Given enough eyeballs, all bugs are shallow" or more formally "Given a large enough beta-testers and co-developers base, almost every problem will be characterized quickly and the problem will be obvious to someone". The name was given in honor of Linus Torvalds (If you are not familiar with the name please Google it NOW!). (Wikipedia)

Now you are probably thinking: “I know what you are trying to say but...”!

Reading your mind, I would continue your thoughts with the upcoming phrase: “...But how can Apple benefit from open sourcing Swift and, moreover, how could this bring more money?”

Exactly! The answer would be by combining the terms mentioned above. By generating more revenue using the “Halo Effect” and “getting things done” faster based on Linus’s Law.

Does this actually mean that Swift will be open sourced based on this combination?

Better security and design of the programming language: More people will be able to see and test the code. The result will be more flaws will be found and fixed faster. From a business perspective, Apple could cut down its development costs and at the same time speed up the delivery of new products. Saving money means also making more money. All community members provide free R&D (Research & Development) back to the center of the community as a whole.

More trust and even more quality: The number of highly skilled developers contributing to the open source project of Swift grows day by day. That means better quality. Also you are able to see for yourself the core language code and be more confident in what is going on. Going once again to the business perspective, those developers create huge global referral networks and, based on the Halo Effect, Apple gains a better reputation, which brings more shares on the market being created by the open sourced technology. Good reputation and more shares could mean more money

from services, like support, documentation and tutorials.

Freedom and concentrated power to the developers: If we were able to get a glimpse into the future, Swift might be a one for everything programming language for the ones who master it. That means it will give them the freedom to create their back-end and front-end systems using one technology. Also, cross platform software could be native for each platform being seen as a separate software entity. And no! Java cannot do that. Remember, at this point, the overall impression of Apple to people and especially developers. Companies could use fewer developers for more of their needs being influenced by Apple's company and brand name which goes alongside with Swift. Could you count the shares on the market that will be created for Apple once more? At least try to imagine it!

Already, developers can use the open source version of Swift and create applications on Linux. Also there are implementations of server side development using Swift.

By the time I am writing this, I am convinced that the future of Swift is bright. In general, as an overview, the future of open source technologies is bright. Could open source be a business as a whole? Could corporations and individual professionals make profit from open source? On both questions we would say YES! Definitely YES!

Shared economy! Open innovation! Open data! Open source might be the new business era!



About the Author:

Georgios is an Indie iOS Developer and also the Director of Pessios Ltd., a small development studio based in London, UK. He holds a Masters degree in Electrical and Computer Engineering. During his professional career as a software engineer, Georgios has worked for start ups and big corporations, like SAP, as a front end developer and mobile applications developer, as well. At the moment, mastering Apple's Swift language and contributing to open source projects are his main

focus. You could reach him on the following links:

Website: <http://gpessios.com/>

LinkedIn: <https://www.linkedin.com/in/gpessios>

Twitter: <https://twitter.com/gpessios>

GitHub: <https://github.com/gpessios>

Memory Management in Swift

by Arash Z. Jahangiri

Managing how long an object is going to stay alive in memory, before its memory is disposed, is called "Life Cycle Management". In some languages, like Java, this is done automatically. That is, GC (Garbage Collector) is responsible for releasing the objects that are no longer used from memory, in order to prevent a memory leak. As a result, the memory is disposed and is ready to be used by other objects if needed. However, in Swift, ARC (Automatic Reference Counting) does this task by keeping track of memory, and releasing the memory as soon as it finds an object with no references to it.

GC happens in the background, whereas in ARC no background processing is done, which makes it more efficient on lower-power systems such as mobile devices.

Swift uses Automatic Reference Counting (ARC) to track and manage your app's memory usage. "In computer programming, tracing garbage collection is a form of automatic memory management that consists of determining which objects should be deallocated by tracing which objects are reachable by a chain of references from certain "root" objects, and considering the rest as "garbage" and collecting them". In Swift, you do not need to think about memory management yourself; ARC automatically frees up the memory used by class instances when those instances are no longer needed.

GC	ARC
<ul style="list-style-type: none"> • GC can clean up entire object graphs, including retain cycles. • GC happens in the background, so less memory management work is done as part of the regular application flow. • Because GC happens in the background, the exact time frame for object releases is undetermined. • When a GC happens, other threads in the application may be temporarily put on hold. 	<ul style="list-style-type: none"> • Real-time, deterministic destruction of objects as they become unused. • No background processing, which makes it more efficient on lower-power systems, such as mobile devices. • Cannot cope with retain cycles.

Figure 1. Comparison of GC and ARC.

However, in some cases, ARC needs more information about your code in order to manage memory for you. In the following, we are going to provide some examples for these situations.

An Introduction to ARC

Whenever you create a new instance of a class, ARC allocates a piece of memory to it, this to store all the information about this instance. When that instance is no longer needed, ARC frees up this piece of memory, in order to be used for other purposes. ARC needs to do this task alertly, since if it deallocates an instance which is still in use, you cannot access that instance anymore; otherwise, your app would crash.

So, ARC uses Reference Counting in order to track how many variables are referring to a specific instance of a class. That is, ARC deallocates the instance as soon as no active reference to that instance exists.

An Example: How ARC works

We define a class named: Pet, which has a property called name. This class also has an initializer and a deinitializer.

```
class Pet {  
  
    let name: String  
    init(name: String) { self.name = name  
  
    print("\(name) is being initialized")  
  
    }  
  
    deinit {  
  
    print("\(name) is being deinitialized")  
  
    }  
  
}
```

Now, we define three variables of type `Pet?`, which are going to refer to the same instance of `Pet` class.

```
var reference1: Pet?  
  
var reference1: Pet?  
  
var reference1: Pet?
```

We are going to create a new `Pet` instance and assign it to one of these variables:

```
reference1 = Pet(name: "Kelly")  
  
// prints "Kelly is being initialized"
```



Figure 2. New Pet instance.

Because the new Pet instance is assigned to reference1, there is a strong reference from reference1 to this new Pet instance. As a result, ARC makes sure it is not deallocated.

If we assign this new instance to the other variables as well, we will have three references to the same Pet instance:

```
reference2 = reference1
reference3 = reference1
```

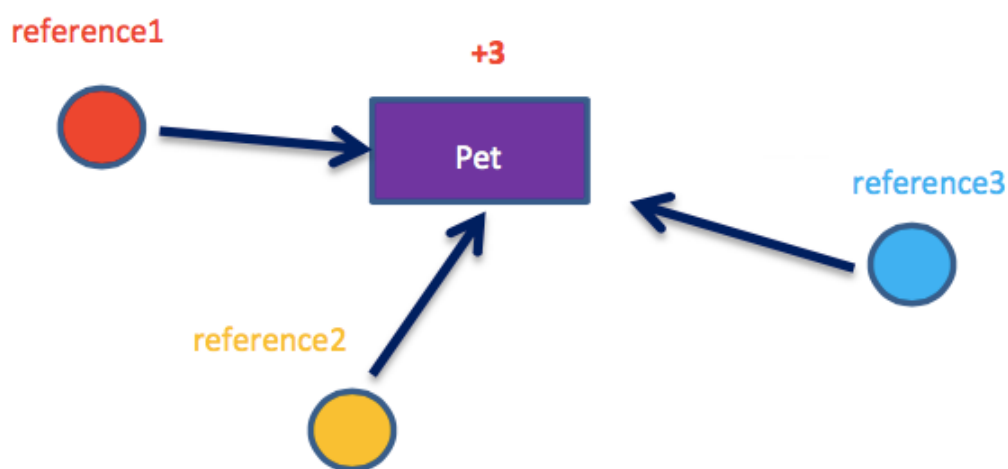


Figure 3. Three references to the Pet instance.

If we break two of these strong references (including the original reference) by assigning nil to them, there will remain one strong reference, so ARC would not deallocate the Pet instance:

```
reference1 = nil
reference2 = nil
```

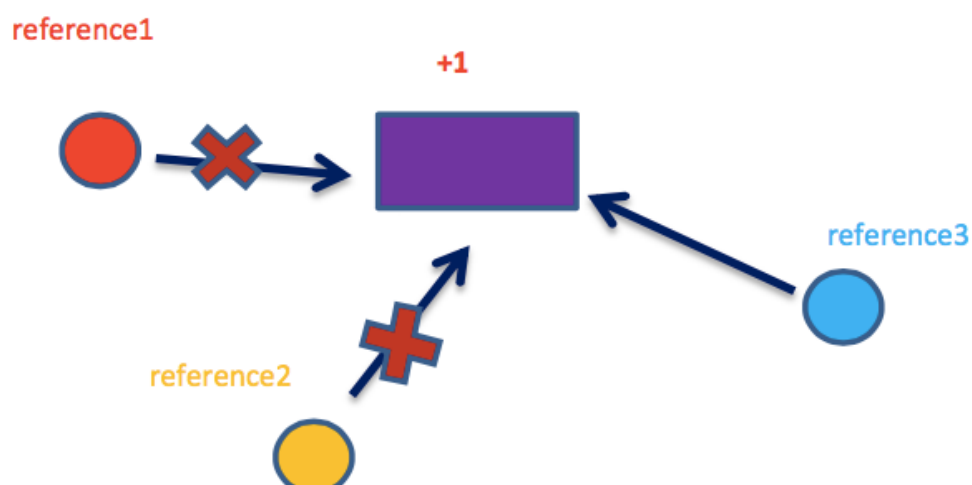


Figure 4. ARC not deallocating the Pet instance.

ARC will not deallocate this Pet instance, until the third reference - reference3 - is broken. At that time it is obvious that the instance is no longer in use and deallocating it by ARC will not result in error.

```
reference3 = nil

// prints "Kelly is being deinitialized"
```

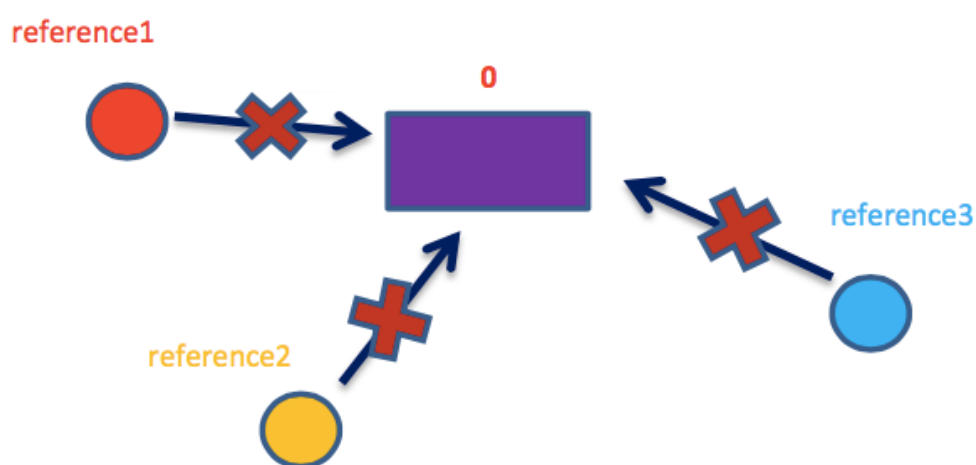


Figure 5. The instance is no longer in use and deallocating it by ARC.

Strong Reference Cycles Between Class Instances

In some situations, a class never gets to the point where it has zero strong references. This can happen if two class instances hold a strong reference to each other, such that each instance keeps the other alive. This is known as a strong reference cycle.

Here is an example of how a strong reference cycle can be created by accident.

Every Person instance has a name property of type String and an optional car property that is initially nil. The car property is optional, because a person may not always have a car.

```
class Person { let name: String

init(name: String) { self.name = name } var Car: car?

deinit { print("\(name) is being deinitialized") }

}
```

```
class Car {
    let make: String
    init(make: String) { self.make = make } var owner: Person?
    deinit { print("Car \(make) is being deinitialized") }
}
```

Similarly, every Car instance has a make property of type String and has an optional owner property that is initially nil. The owner property is optional because a car may not always have an owner.

Now, we define two variables of optional type called: Bernie and Ford.

```
var Bernie: Person?
var Ford: Car?
```

We can now create a specific Person instance and Car instance and assign them to these variables:

```
Bernie = Person(name: "Bernie")
Ford = Car(make: "Ford")
```

As you see in the figure, Bernie has a strong reference to the new Person instance, and Ford has a strong reference to the new Car instance:

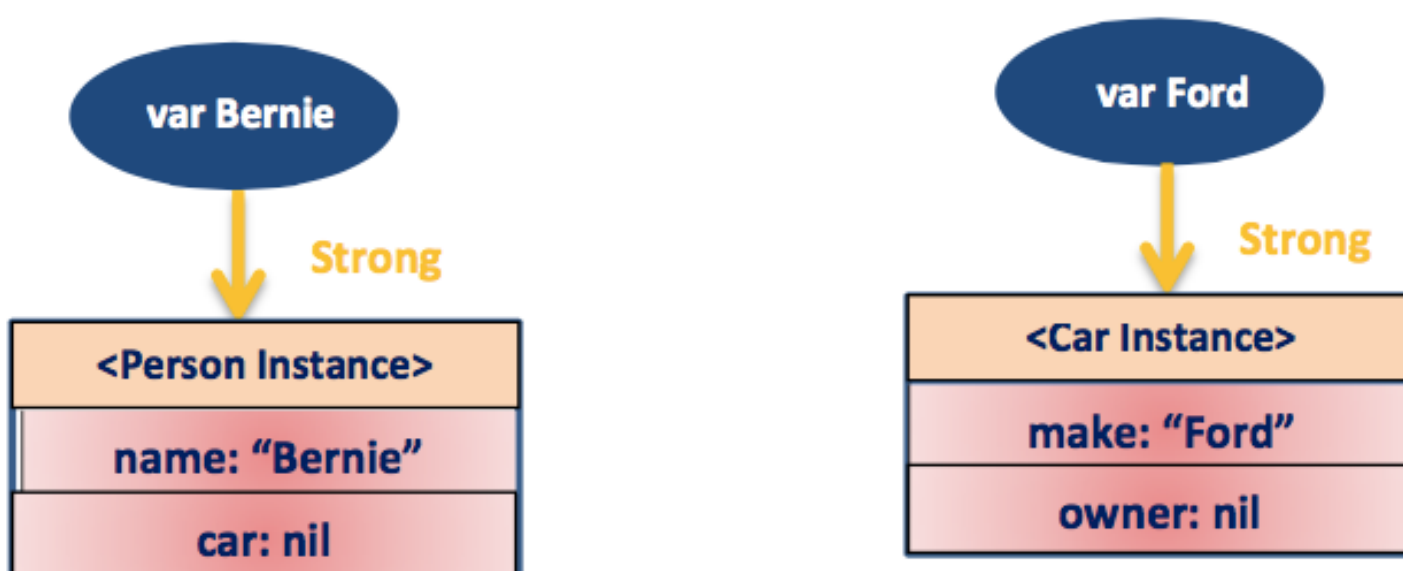


Figure 6. Bernies and Fords references.

We can now link the two instances together so that the person has a pet, and the pet has an owner:

```
Bernie!.car = Ford
Ford!.owner = Bernie
```

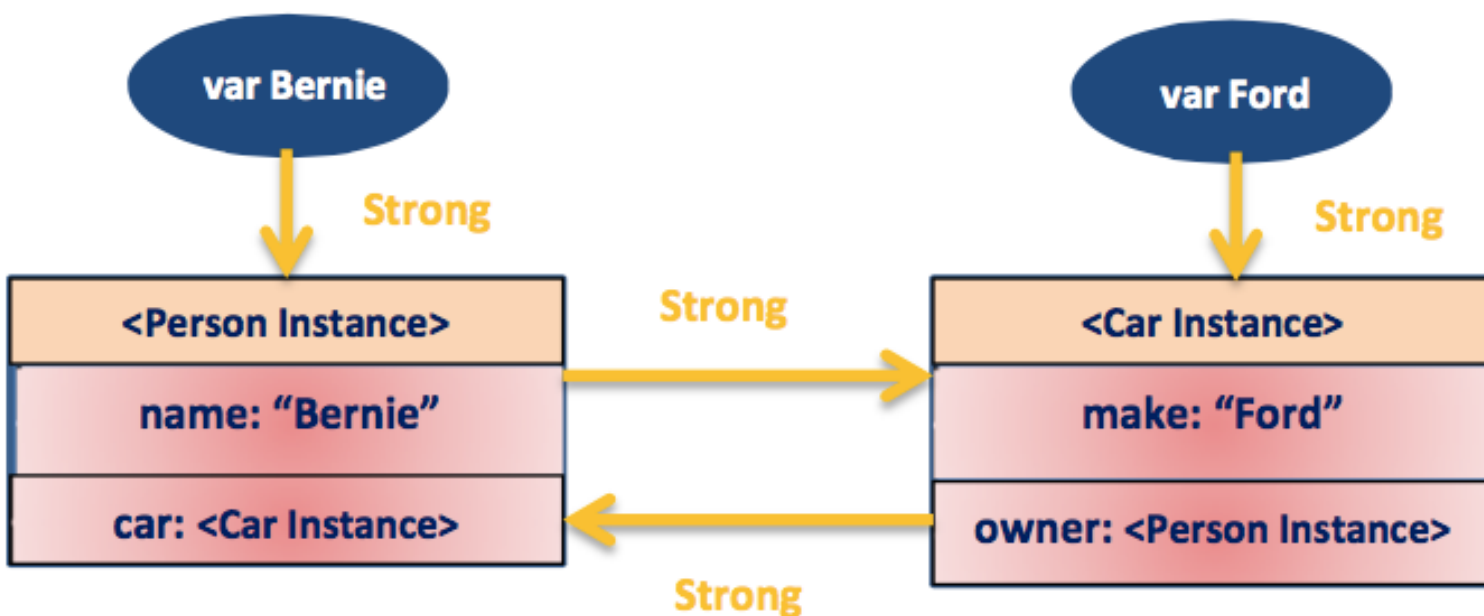


Figure 7. Bernies and Fords references.

Unfortunately, linking these two instances creates a strong reference cycle between them. The Person instance now has a strong reference to the Car instance, and the Car instance has a the Bernie and Ford variables, the reference counts do not drop to zero, and the instances are not deallocated by ARC:

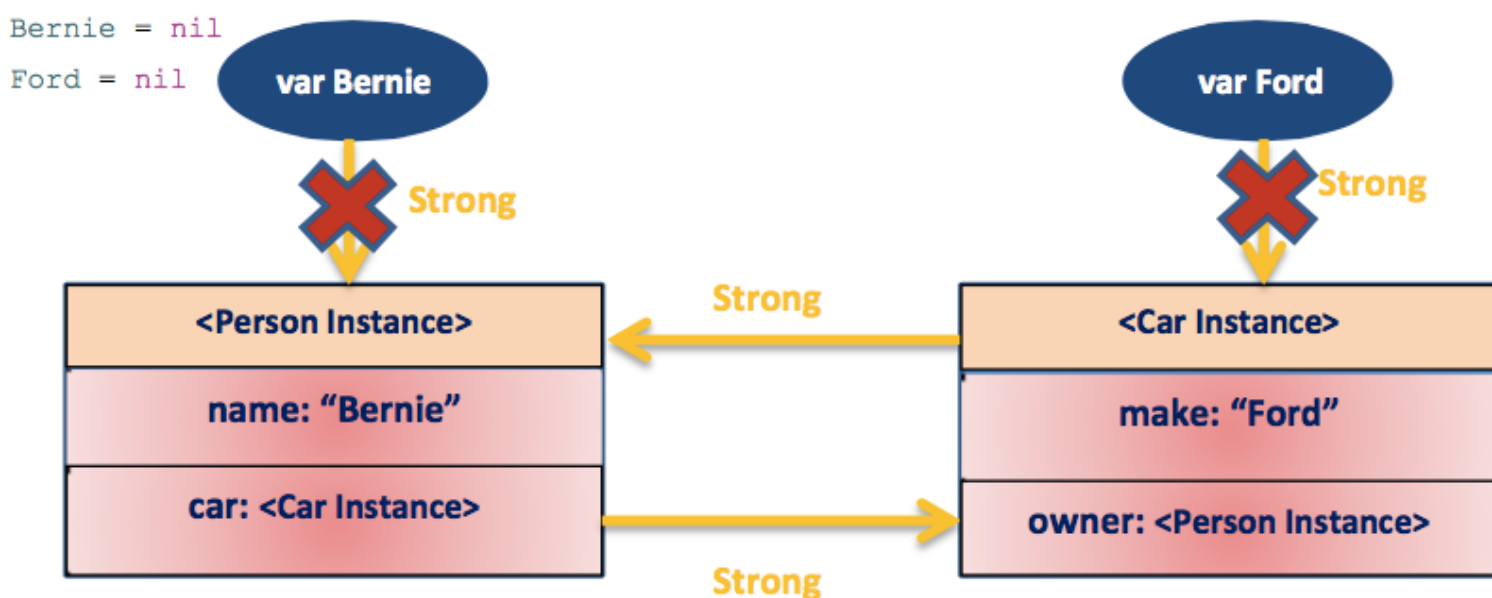


Figure 8. Not deallocated instances by ARC.

You see that neither deinitializers were called when you set these two variables to nil. The strong reference cycle prevents the Person and Car instances from ever being deallocated; thus causing a memory leak in your app.

The strong references between the Person instance and the Car instance remain and cannot be broken. This problem is called Strong Reference Cycle. We are going to introduce two solutions for that.

Resolving Strong Reference Cycles Between Class Instances

Swift provides two ways to resolve strong reference cycles: weak references and unowned references.

Weak and unowned references enable one instance in a reference cycle to refer to the other instance without keeping a strong hold on it. The instances can then refer to each other without creating a strong reference cycle.

Use a weak reference whenever it is valid for that reference to become nil at some point during its lifetime. Conversely, use an unowned reference when you know that the reference will never be nil once it has been set during initialization.

First Solution: Weak Reference

By using a weak reference instead of a strong reference, you let ARC dispose the referenced instance. You place the Weak keyword while declaring a property.

Weak references are allowed to have nil value, so you must declare all weak references as an Optional type.

Now we are going to declare the Car type's owner property as a weak reference:

```
class Person { let name: String

init(name: String) { self.name = name } var Car: car?

deinit { print("\(name) is being deinitialized") }
```

```
class Car {
    let make: String

    init(make: String) { self.make = make } weak var owner: Person?

    deinit { print("Car \(make) is being deinitialized") }
}
```

The instances are defined as before:

```
var Bernie: Person? var Ford: Car?

Bernie = Person(name: "Bernie")

Ford = Car(make: "Ford")

Bernie!.car = Ford

Ford!.owner = Bernie
```

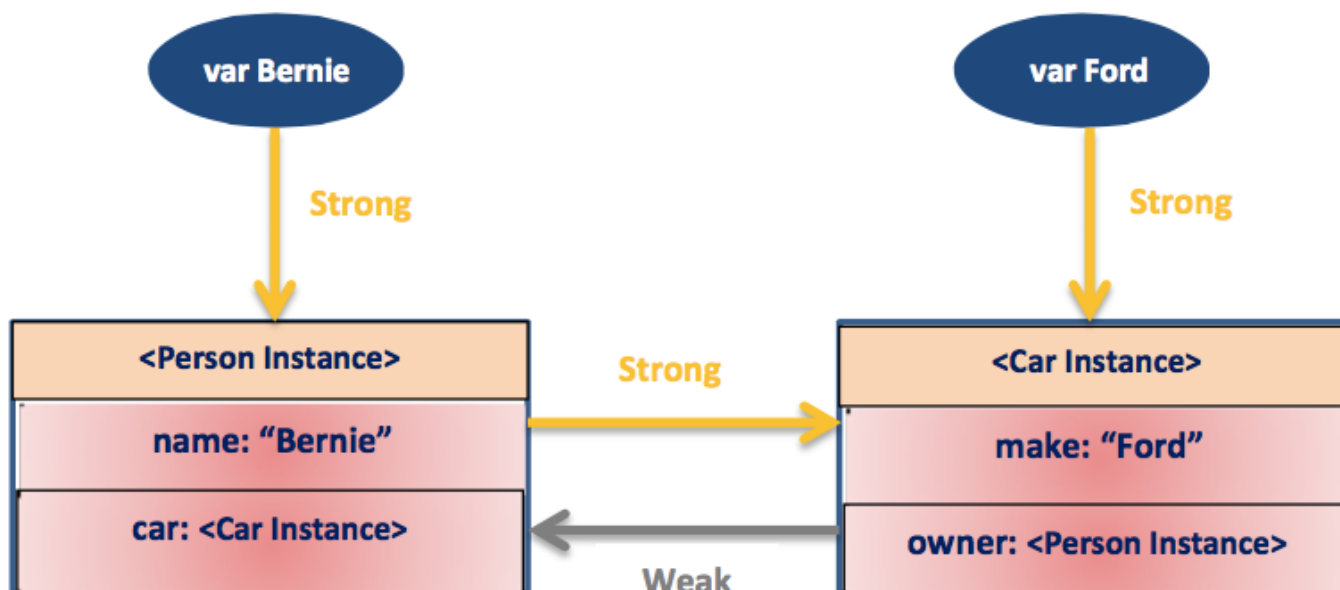


Figure 9. Declaring the Car type's owner property as a weak reference.

Now if we break the strong reference held by the variable Bernie, there would be no more strong references to this Person's instance:

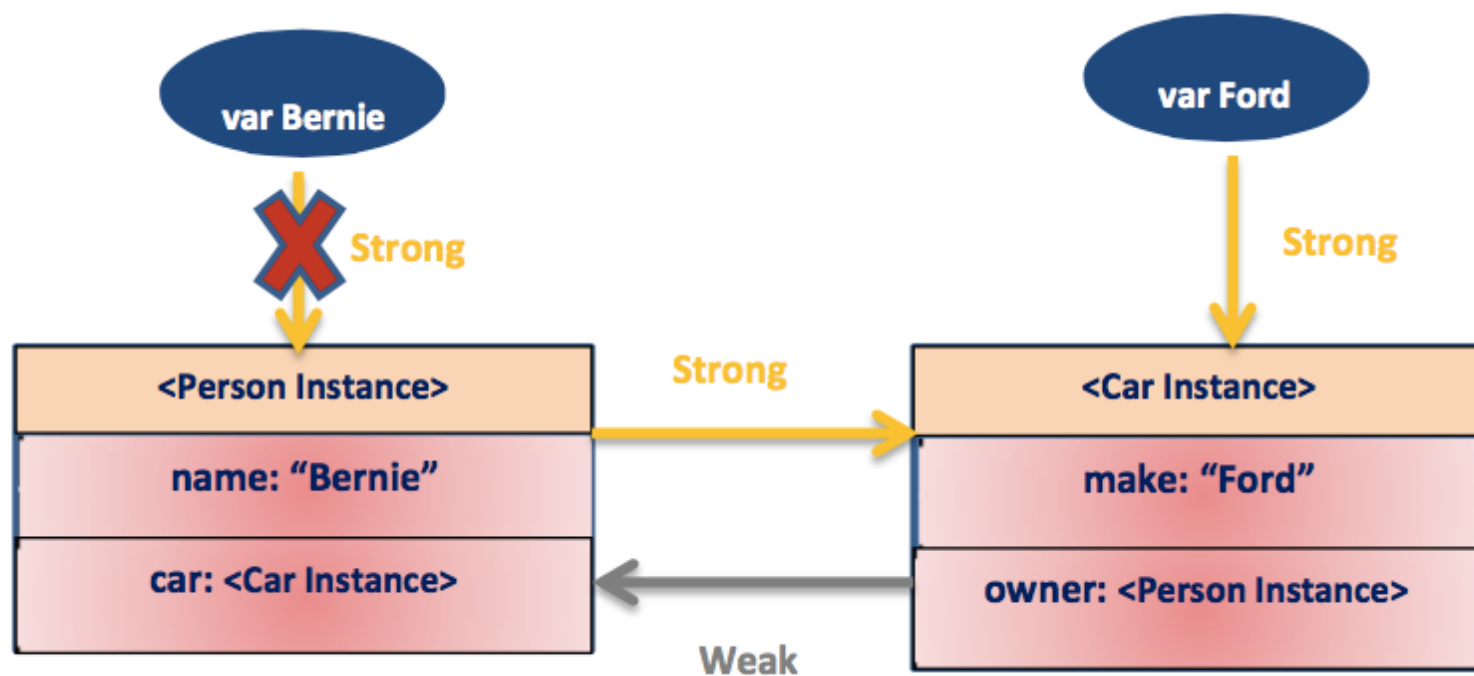


Figure 10. Breaking strong references.

As a result, ARC will dispose this Person instance:

```
Bernie = nil
// prints "Bernie is being deinitialized"
```

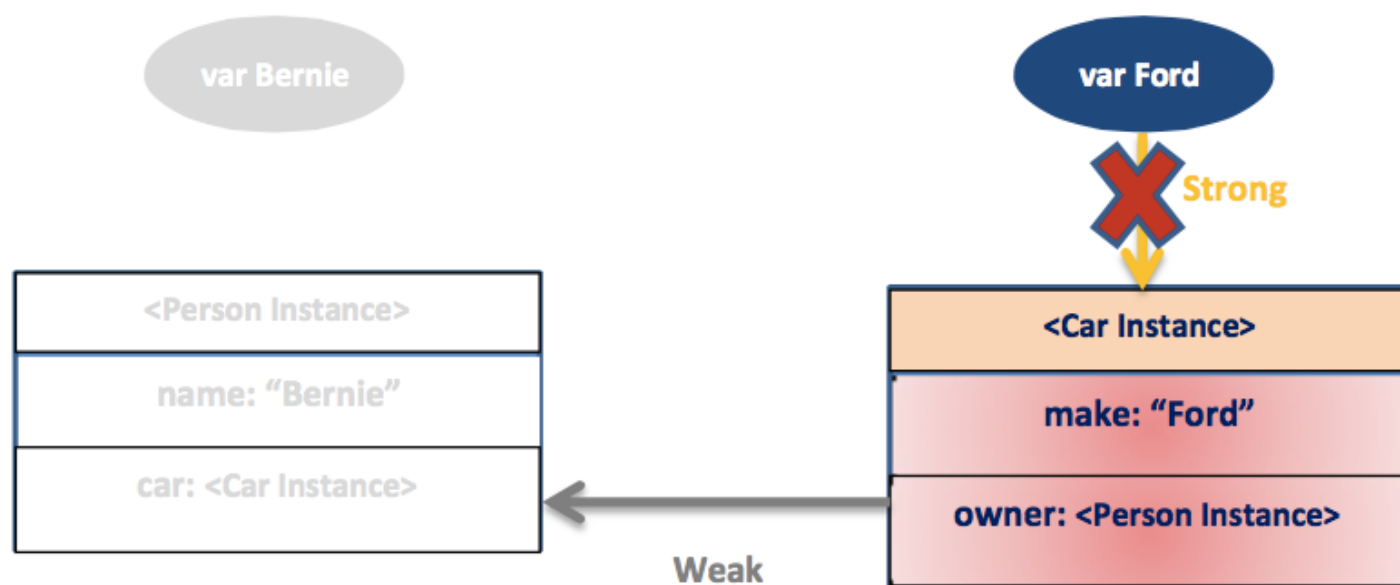


Figure 11. ARC disposing Persons instance.

When/if we break the reference to the Car instance from the Ford variable, ARC will deallocate this instance as well, since there are no strong references to this Car instance:

```
Ford = nil

// prints "Ford is being deinitialized"
```

Second Solution: Unowned References

An unowned reference is assumed to always have a value, and it can never be nil like a weak reference. As a result, an unowned reference is always defined as a non-optional type. We can place unowned keyword before a variable name in order to declare it as an unowned reference.

As an example we have defined two classes called Customer and CreditCard:

```
class Customer { let name: String

var card: CreditCard? init(name: String) { self.name = name
}

deinit { print("\(name) is being deinitialized") }
}

class CreditCard { let number: UInt64

unowned let customer: Customer init(number: UInt64, customer: Customer) { self.number = number

self.customer = customer
}

deinit { print("Card #\(number) is being deinitialized") }
}
```

The relationship between these two classes might create a strong reference cycle. In this example, a customer may or may not have a CreditCard, but a CreditCard is always be linked to a customer. So, in Customer class, we defined CreditCard as an optional property in Customer class, but the CreditCard class has a non-optional property called customer. Because a CreditCard will always have a customer, we defined its customer property as an unowned reference.

Now we define a customer:

```
var Fred: Customer?

Fred = Customer(name: "Fred")

Fred!.card = CreditCard(number: 1234_5678_9123_4567, customer: Fred!)
```

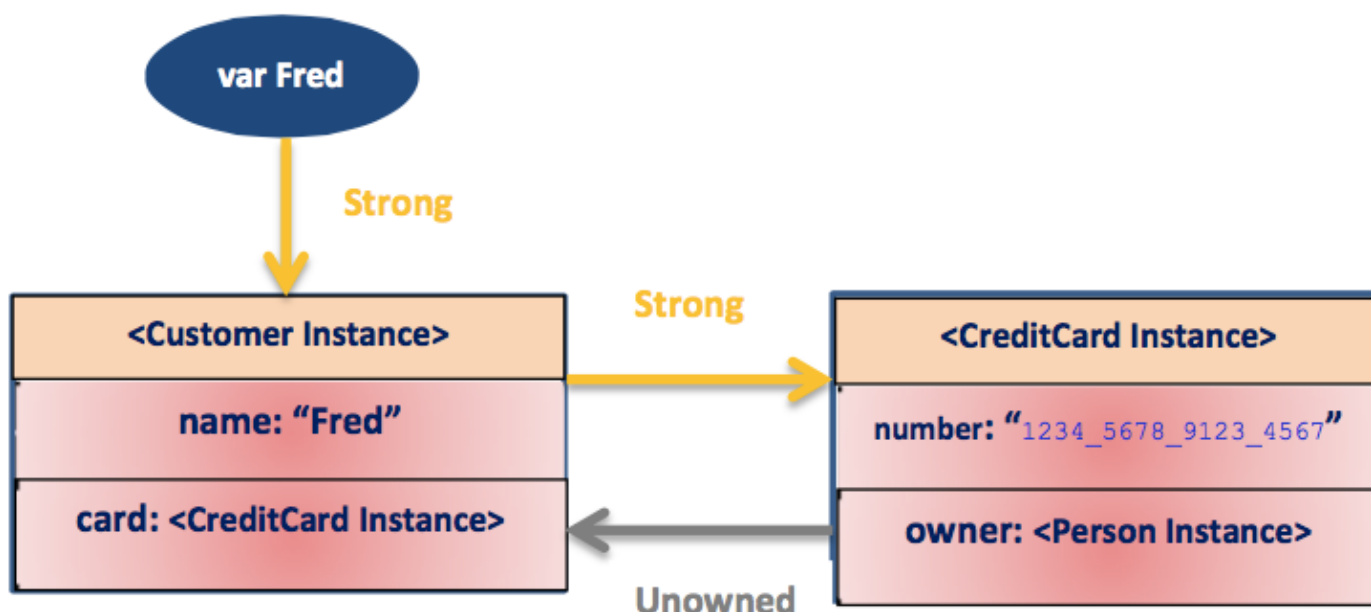


Figure 12. Defining a customer.

As you see, there is always a strong reference in this example. So, when we break the strong reference from Fred to this customer instance, ARC will dispose it.

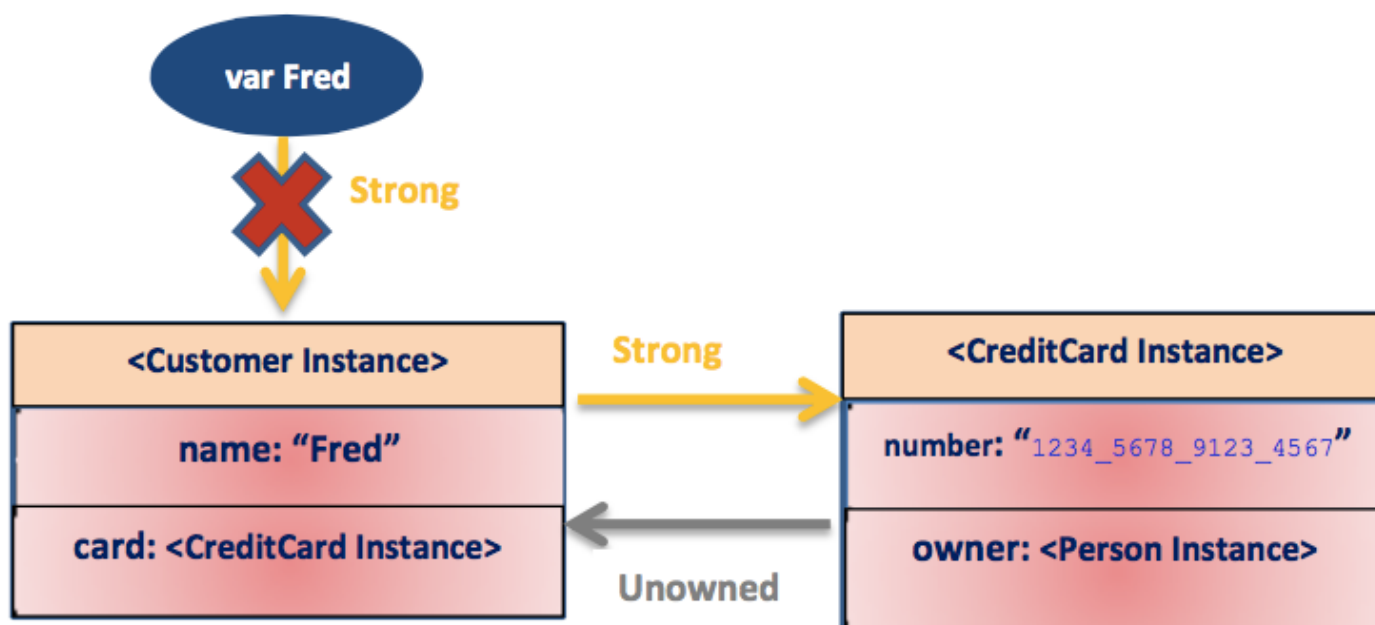


Figure 13. ARC disposing strong reference form Fred.

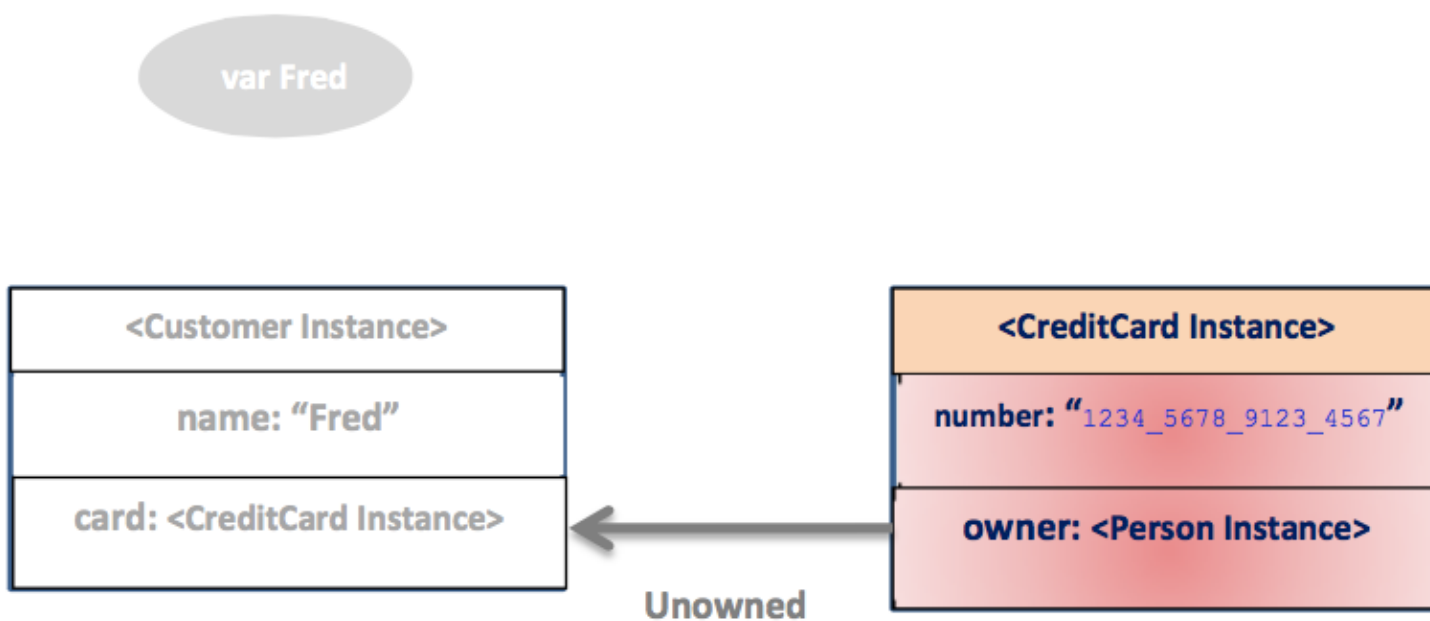


Figure 13. ARC disposing strong reference form Fred (part 2 of the figure 13).

As a result, since there are no strong references to the `CreditCard` instance, ARC will deallocate it as well.

```
Fred = nil

// prints "Fred is being deinitialized"

// prints "Card #1234567891234567 is being deinitialized"
```

Conclusion

In this paper, we introduced ARC action in Swift. We learned that ARC manages memory for us; however, in a few cases ARC requires more information about our code in order to manage our app's memory usage. In fact, in some situations, an instance of a class never gets to a point where it has zero strong references. This is known as a "strong reference cycle". We provided two solutions for this issue. By utilizing these solutions, we intend to enable ARC to manage all of our app's memory.

References:

www.developer.apple.com



About the Author:

Arash, currently is Lead iOS developer at yarima Co. He's focus is mainly on Healthcare projects. He is passionate about creating well-crafted code and teaching others.

Outside of coding, he enjoys spending time in nature with his wife, and playing Trumpet.

twitter: [ichbinarash](https://twitter.com/ichbinarash)

linkedin: <https://www.linkedin.com/in/arash-z-jahangiri-b4865976>

Why FreeBSD?

by Hamza Sheikh

Before going into why FreeBSD is now my preferred OS for learning UNIX, let us review why I used Linux for a long time.

TEASER END: Read more

I started with Red Hat (before RHEL) in the early 2000s; tried to install it on a variety of hardware from CDs found in the back pages of books from the library. There was not a single working install so I gave up. Then came the Ubuntu hype, and around 2006 I built a computer specifically to install and learn it. Everything - except Wi-Fi - worked out of the box. I managed to get Wi-Fi working with ndiswrapper reading a lot of community documentation. I now had a working box and I began my Linux journey.

My progress from using Linux at home, to developing stuff at work, was rapid and made easy by the entire free and open source community. As Matthew Garrett puts it:

“I am who I am because people made the choice to release their software under licenses that permitted examination, modification and redistribution. I am who I am because I was able to participate in communities that took advantages of those free-

doms to produce new and better software”.

There were many rough edges in the Linux world - some of them exist even today. Choosing the right distribution (distro) for the task at hand is always the first and most difficult decision to make. While this is a strength of the Linux community, it is also its weakness. This is exacerbated with the toxic infighting within the community in the last few years.

A herd of voices believes it is their right to bring down a distro community because it is not like their distro of choice. Forking upstream projects has somehow become taboo. Hurling abuse in mailing lists is acceptable. Helping new users is limited to lambasting their distro of choice. Creating conspiracy theories over software decisions is the way to go. Copyleft zealots roam social media declaring non-copyleft free software heretic abominations. It all boils down to an ecosystem soured by the presence of maniacs who have the loudest voices and they seem to be everywhere you turn.

Where is the engineering among all this noise? Btrfs - baking for a long time - is still nowhere near ZFS in stability or feature parity. systemd is an insatiable entity that feeds on every idea in sight and just devours indiscriminately. Wayland was promised years ago and its time has yet to arrive. Containers are represented by Docker that neither securely contains applications nor makes them easy to manage in production. Firewalling is dithering between firewalld, nftables, etc. SystemTap cannot match DTrace.

In the same time span, what do various BSDs offer? pf, CARP, ZFS, Hammer, OpenSSH, jails, pkgsrc, (software) ports, DTrace, hardware portability; just to name a few. Few would deny that BSDs have delivered great engineering with free software licenses to the entire world. To me they appear to be better flag bearers of free software with engineering to back it.

That was my initial motivation to start with FreeBSD. I was enticed to the good side of the force first by Allan Jude and Kris Moore through their BSD Now podcast and then by Bryan Cantrill's interviews (Ubuntu Slaughters Kittens, The Cantrill Strikes Back) and talk (Fork Yeah! The Rise and Development of illumos). Why I stayed is the crux of this post.

Choosing between DragonFly BSD, FreeBSD, NetBSD, and OpenBSD involves a few subjective thoughts. The only hardware available to me was a Raspberry Pi 2 (RPi2). Any BSD I chose had to work on it. That eliminated DragonFly BSD and OpenBSD from the get go.

I have great respect for the NetBSD project as they claim and aim to be ultra-portable. It

works on RPi2 plus many other SOCs (System on a Chip). This is a great advantage because any user would need to learn one OS and run it on a multitude of available hardware.

Pkgsrc is an ambitious and well executed project and I admire the engineering work behind it. That Minix and SmartOS both use pkgsrc instead of rolling their own packaging system testifies to its capabilities. It even has Erlang and Go available which are on my list of things to learn. Too bad it does not have Rust and Elixir available; however, I expect it will soon.

NetBSD, unfortunately, is missing a couple things I wanted to try: namely jails and bhyve. I realize bhyve does not work on RPi2. In addition, I had found RaspBSD which installed beautifully on RPi2. My couple attempts at installing NetBSD in a VM, with its capable but minimal installer, were a far cry from the FreeBSD installer. It also does not have marketing making a case for it in the wider community; that lays a few seeds of doubt in my mind about its long-term viability. Lastly, Digital Ocean does not support it out of the box and I wanted to keep short the number of OSs I have to learn simultaneously. For these reasons I kept NetBSD as an alternative choice if FreeBSD did not work out.

I like to think of FreeBSD as the Debian of BSDs. It has a much bigger tent than others that results in more features available to its users. As I have found on the FreeBSD ARM mailing list, its community is super helpful and friendly to new users. The community appears to be more open minded than say OpenBSD which makes a lot of difference.

My RPi2 kit worked out of the box on FreeBSD 11.0-CURRENT, including Wi-Fi. ARM support is improving every day. It is still a Tier-2 supported architecture, but I am optimistic for the future.

I cannot emphasize enough on how much I love the separation between base and ports. Unlike any other Linux distro I have ever used, third party applications get updates independent of the base OS. So the same ports are installable on 9.x-RELEASE, 10.x-RELEASE, and 11.0-CURRENT. I am not bound to the same version of Erlang until the next OS major release the way I am on Debian or Fedora. This freedom is truly liberating.

A ton of packages are available in ports. Recently I wanted to install Erlang on ARM and a pre-built binary package was not available through pkgng. I built it from source doing nothing more than

```
cd /usr/ports/lang/erlang && make  
install clean.
```

Having used Poudriere to build my own set of packages I cannot praise it enough. I wish something like that was available on Debian and CentOS. Pkgng provides binary packages so I do not have to build from source anymore (most of the time). There is on-going work to provide updates to base using pkgng. This attests to the forward progress FreeBSD makes every day.

Jails have been available for more than a decade and things just work. Docker support is also in the works using Jails and ZFS.

ZFS is available as a first class citizen. Although I have never needed to use it to date; the fact that when I do there is a history of people using it in production. I can leverage lessons learned and use a stable product with unmatched features.

Bhyve is a project to keep up with the times. It can run major Linux distros, Windows Server, and BSDs. A native hypervisor was sorely missing and now it is not. Who says BSDs are dead?

If there ever was love at first sight for me in firewall technology it was pf. I picked it up in a couple hours and got enough done that it is running on my FreeBSD VPS. I have not used IPFW but it is available as an alternative and is no slouch either.

Using

```
/etc/rc.conf
```

to configure everything is a much more pleasant experience. No more hunting through a plethora of commands - service, systemctl, chkconfig - to get status or perform actions.

FreeBSD is not perfect. I do not think I will recommend it to someone to use it on the desktop today. There is a lot of progress made by PC-BSD to make that possible. It is still missing a lot of niceties provided by other OSes. Support for bleeding-edge hardware lags because of the small size of the developer community and cursory - if any - support from upstream vendors. With alternatives such as OS X - a capable UNIX on its own - at least I do not feel compelled to use FreeBSD on the desktop. Hopefully this will change

Finally - and I maybe repeating myself here - I have nothing but praise for the community. Be it BSD Now, mailing lists, Reddit, Twitter, LFNW, or SeaGL, people have encouraged me, answered my questions, and filed bugs for me. I have been welcomed and made a part of the community with open arms. These reasons are (good) enough for me to use FreeBSD and contribute to it.

BSD Now

<http://www.bsdnow.tv>

Ubuntu Slaughters Kittens

http://www.bsdnow.tv/episodes/2015_08_19-ubuntu_slaughters_kittens

The Cantrill Strikes Back

http://www.bsdnow.tv/episodes/2015_11_23-the_cantrill_strikes_back

Fork Yeah! The Rise

<https://www.youtube.com/watch?v=-zRN7XLCRhc>

Development of illumos

<https://www.youtube.com/watch?v=-zRN7XLCRhc>

About the Author:

A Linux escapee and FreeBSD user of less than a year. Started using BSDs with pfSense in 2007 and loved it. Day job involves a lot of Python and test automation. Evenings and weekends are devoted to family, FreeBSD, and Erlang.

@aikchar

Samhain - Free, Open Source File Integrity Monitoring / HIDS

by Leonardo Neves Bernardo

What will you learn?

How to use the samhain software to monitor your UNIX operating system

What should you know?

Basics of the UNIX Operating System

Basics of IT Security

In this article we will learn how to use the Samhain software to monitor activities in the UNIX operating system, above all, to monitor file modifications.

One of the defining features of UNIX is 'Everything is a file', and one of the defining features of its administration is 'You cannot manage what you do not measure'. So, if we synthesize these phrases, we can conclude that 'It is necessary to measure our files to be a UNIX administrator'. Well, let us exchange the term measure for monitor, because in an operating system we can 'measure' a lot of features related to files. For example, we can monitor content, owner, and MAC times. The conclusion is 'We can be good UNIX administrators only if we can monitor files'.

Why Samhain?

Samhain is not a simple file integrity checker. It can monitor logs, login/logoff events, open ports, integrity of the kernel, process, and mount points. This is the 'client' part of Samhain, because it can be a network service using Yule and we can monitor everything in a centralized way. There is Beltane as well, the web console for Yule/Samhain. The last interesting feature of Samhain is that it can be installed even on a windows operating system to monitor files and registry keys.

Samhain is a HIDS, or Host Intrusion Detection System, and we have some other free and commercial options available like Aide, Tripwire, Osiris, and Ossec. First of all, you need to know that these tools have advantages and disadvantages; my recommendation is that you install and test all of them before you choose when it is the right time to use any of them. In my experience, Samhain is the best option in most cases, particularly when the major necessity is to check file changes. So, let us look at these tools:

Aide – Aide is only a file integrity check that can be used in a standalone way. It is too simple and probably it is not the best option to use in a large or critical environment. Sometimes your requirement is very specific and you need to check a bunch of files in batch mode. Only in this type of case can Aide be useful for you.

Tripwire – Tripwire has two versions: free and commercial. Tripwire, like Samhain, is a very good file integrity check as well. The free version has less features than the commercial version and both versions do not do what Samhain can do. I consider Tripwire a good option if you are working in a company which requires that only commercial versions can be installed in their environment.

Osiris – Osiris is another option to consider, it is a free and centralized tool like Samhain. Osiris is not as secure as Samhain, because there is no protection for configuration files and databases. Note: The development of this tool seems to have stopped at this moment.

Ossec – I think Ossec is an amazing tool. The way that it can evaluate log files is very good. Unfortunately, Ossec is not so good when it needs to do a file integrity check. These checks are what Samhain does better. I believe Samhain can work with Ossec to create a very robust and complete UNIX HIDS.

Samhain – Lets now look at Samhain and see some disadvantages of this tool; however, there is no perfect tool. Firstly, Samhain has the steepest learning curve of all the tools that we have reviewed so far. Secondly, the log monitoring of Samhain is not very good; Ossec and Ossim do this task much better. As we will see in this article, there are many advantages of using Samhain as opposed to using the other tools discussed above.

Samhain

Installation

Let us do a basic installation of Samhain to see what it can do for us. Download the latest version of the software at <http://www.la-samhna.de/> and follow the next commands:

```
$ ls -l samhain-current.tar.gz
-rw-rw-r-- 1 neves neves 2165891 Feb  9 17:37 samhain-current.tar.gz
$ gunzip samhain-current.tar.gz
$ tar -xf samhain-current.tar
$ ls -l samhain-4*
-rw-rw-r-- 1 neves neves 2168448 Dez 21 15:57 samhain-4.1.2.tar.gz
-rw-rw-r-- 1 neves neves      181 Dez 21 15:58
samhain-4.1.2.tar.gz.asc
$ tar -xf samhain-4.1.2.tar.gz
$ cd samhain-4.1.2
$ ./configure && make
$ sudo make install
$ sudo make install-boot
```

When installing Samhain, make sure that directories like /var/, /var/lib and /var/log are owned by root before the installation as Samhain does not like to use directories that other users can change.

Instead of using 'make && make install', you might prefer to create a package for your operating system. The options to do this are:

```
make rpm|deb|tbz2|depot|solaris-pkg
```

It is important to note that our installation was very simple, in fact we used the default installation, and the configure and make have a lot of options.

Running

The first step after we install is to initialize the database for file signatures. The path to the database is compiled in, the default path is

```
/var/lib/samhain/samhain_file  
  
# samhain -t init
```

Samhain will spend some minutes to create `samhain_file`. A `samhain_file` is like a picture of the attributes of the files of your entire system. In my desktop the file created has 384MB:

```
# ls -lh /var/lib/samhain/samhain_file  
  
-rw-r----- 1 root root 348M Feb  9 19:59 /var/lib/samhain/  
samhain_file
```

Let us look inside the file `samhain_file`. My file has around of 2.5 million lines, let us now get the information about the file `/etc/passwd`:

```
# egrep --binary-files=text '/etc/passwd$' /var/lib/samhain/  
samhain_file  
  
❖KV-----rootoot❖-rw-r--  
r--86F8243B317A508CA3733569DAE0061C1BD4A4C9702FAA82/etc/passwd
```

As you can see, this file is binary, but it is not encrypted. It is possible to see for example that the mode of this file is `-rw-r--r--`, and that the owner and the group are `root`. Other interesting information is the string `'86F8243B317A508CA3733569DAE0061C1BD4A4C9702FAA82'`, it is the TIGER192 Digest of the file. TIGER192 is a hash function, similar to MD5 and SHA. This information is necessary inside the `samhain_file` because when some file is changed, Samhain will run TIGER192 again and compare the results. If TIGER192 differs from the new calculated value, Samhain will trigger an alert.

Let us now compare the information inside `samhain_file` and what we can see using `'stat'` tool:

```
# stat /etc/passwd  
  
File: '/etc/passwd'
```

Samhain

```
Size: 2239           Blocks: 8           IO Block: 4096   regular
file

Device: 804h/2052d   Inode: 2765120   Links: 1

Access: (0644/-rw-r--r--)  Uid: (    0/   root)   Gid: (    0/   root)

Access: 2016-02-09 20:11:35.443634355 -0200
Modify: 2015-11-17 21:53:49.912451000 -0200
Change: 2015-11-17 21:53:49.948451000 -0200

Birth: -
```

Information about size, inode, access, and modify times and other information are stored in `samhain_file`, even though it is not possible to see using 'grep'.

Now we can start the service:

```
# /etc/init.d/samhain start

ERROR : [2016-02-09T20:58:17-0200] msg=<No matches found>, inter-
face=<glob>, path=</var/lib/rpm/__db.00?>

* Service samhain started
```

As you can see, we have got an ERROR message. The problem is that we did not change the configuration file `/etc/samhainrc` to reflect our system, we are using the default `samhainrc`.

Let us force a change in our file `/etc/passwd` and see what happens:

```
# ls -l /etc/passwd

-rw-r--r-- 1 root root 2239 Feb  9 21:06 /etc/passwd

# chmod g+w /etc/passwd

# ls -l /etc/passwd

-rw-rw-r-- 1 root root 2239 Feb  9 21:06 /etc/passwd
```


Inside `/var/log/samhain_file` we can see the alert about our change:

```
CRIT    :  [2016-02-10T18:52:50-0200] msg=<POLICY [ReadOnly] -----M--
T->, path=</etc/passwd>, mode_old=<-rw-r--r-->, mode_new=<-rw-rw-r-
->, attr_old=<----->, attr_new=<----->,
ctime_old=<[2015-11-17T23:53:49]>, ctime_new=<[2016-02-09T23:14:05]>,
mtime_old=<[2015-11-17T23:53:49]>, mtime_new=<[2016-02-09T23:06:38]>,
```

So, maybe you did not get this message. The problem is that in default configuration, samhain will check changes in filesystems only at intervals of 2 hours. This interval is configured using `SetFileCheckTime` parameters in `/etc/samhainrc`. The default `SetFileCheckTime` in `samhainrc` looks like bellow:

```
# Interval between file checks

SetFileCheckTime = 7200
```

Probably you are thinking that is a good idea to change this parameter and get alerts more frequently. It can be a good or a bad idea, it depends on your system. If you are using Linux, you can take the advantage of `inotify` feature of kernel. First, we need to make sure that the kernel will handle a considerable number of `inotify` watches:

```
# echo 1048576 > /proc/sys/fs/inotify/max_user_watches
```

And we need to be sure that our change is permanent, including in `/etc/sysctl.conf` the following line:

```
fs.inotify.max_user_watches=1048576

Inside /etc/samhainrc, include a section for inotify:

[Inotify]

InotifyActive = yes

InotifyWatches = 1048576
```

When Samhain is using `inotify`, it is not necessary anymore to perform filesystems scans to detect file changes, this because when a file is changed the kernel will notify Samhain about the changes. Both the I/O and CPU loads will be reduced a lot.

Another advantage is that Samhain will report immediately. Remember that our SetFileCheckTime was configured to 7200 seconds, on average we are notified on 1 hour after a change was made, now samhain is notifying in seconds after a file change is made.

Using inotify, we can change SetFilecheckTime to a huge value, like 315360000 (around 10 years). It will prevent samhain doing frequent file scans.

Unfortunately you can use inotify only in Linux systems. In other operating systems you need to find a suitable value for SetFilecheckTime. It will depend on the number of files monitored and the consequent time to do a scan. Sometimes a once a day check is sufficient, on the other hand you might need notification intervals of only a few minutes. It is important that the time of a scan (samhain -t init for example) is smaller than the value configured in SetFilecheckTime.

We have two other important parameters related to performance. If you do not change the default value of SetNiceLevel, Samhain can use too much of your system CPU resources. To avoid this happening, it is a good idea to set nice level to this:

```
SetNiceLevel = 19
```

In the same way, Samhain can use too many I/O resources when you leave SetIOLimit unchanged and you are not using Inotify feature. To limit I/O usage to 100MB/s, it can be set like this:

```
SetIOLimit = 100000
```

SetIOLimit affects the time of filesystem scans as well as the init task (samhain -t init).

Samhain will use a considerable amount of memory because it will maintain the baseline database in memory.

Analyzing the logs

Let us now look at the format of some important Samhain alerts. The most frequent log messages are 'POLICY' messages, like the following:

```
CRIT    : [2016-02-10T18:52:50-0200] msg=<POLICY [ReadOnly] -----M--
T->, path=</etc/passwd>, mode_old=<-rw-r--r-->, mode_new=<-rw-rw-r-
->, attr_old=<----->, attr_new=<----->,
ctime_old=<[2015-11-17T23:53:49]>, ctime_new=<[2016-02-09T23:14:05]>,
mtime_old=<[2015-11-17T23:53:49]>, mtime_new=<[2016-02-09T23:06:38]>,
```

POLICY messages always happen when a file or directory is changed. The name of the policy in this case is [ReadOnly], the absolute path of the file is /etc/passwd, and for every attribute in the file that was changed we have a pair of attribute_old and attribute_new because we need to know what happened with our file. As you can see, in general when we changed an attribute, sometimes other attributes change as a consequence.

In the case above, the file was changed by a chmod, let us see what happens if we change the content of /etc/samhainrc using the following command:

```
echo "" >> /etc/samhainrc
```

As a result, we will get a register in /var/log/samhain_log similar to that:

```
CRIT    : [2016-02-14T10:40:44-0200] msg=<POLICY [ReadOnly] C-----
TS>, path=</etc/samhainrc>, size_old=<15781>, size_new=<15782>,
ctime_old=<[2016-02-14T12:38:53]>, ctime_new=<[2016-02-14T12:40:43]>,
mtime_old=<[2016-02-14T12:38:53]>, mtime_new=<[2016-02-14T12:40:43]>,
chksum_old=<D4F124C86669A5C9D0DE1E0F2F5AD3889EBB4333C8C7F715>,
chksum_new=<3FDDC998EA18391C40265724444DBCCDCCCF7AE0BA4421BD>,
```

As you can see, the change of the size was reported, as well as the change of the checksum (chksum in the log). By default, Samhain reports the change of the content, but it does not report what changed inside the file. It is possible to configure Samhain to store and report the full content of files. There is a limit of 9,200 bytes after zlib compression. To control content of files I strongly recommend some version control software like Subversion.

If you change a monitored file using the Vim editor, the logs resulting will be much more detailed, such as this:

```
CRIT    : [2016-02-14T11:16:03-0200] msg=<POLICY MISSING>, path=</etc-
/4913>
```

```
CRIT    : [2016-02-14T11:16:03-0200] msg=<POLICY MISSING>, path=</etc-
/samhainrc>, mode_old=<-rw----->, attr_old=<----->,
imode_old=<33152>, iattr_old=<0>, hardlinks_old=<1>, idevice_old=<0>,
inode_old=<2753364>, owner_old=<root>, iowner_old=<0>,
group_old=<root>, igroup_old=<0>, size_old=<15782>, size_new=<0>,
ctime_old=<[2016-02-14T12:40:43]>, atime_old=<[2016-02-14T12:38:53]>,
mtime_old=<[2016-02-14T12:40:43]>,
chksum_old=<3FDDC998EA18391C40265724444DBCCDCCCF7AE0BA4421BD>
```

```
CRIT    : [2016-02-14T11:16:04-0200] msg=<POLICY ADDED>, path=</etc-  
/samhainrc>, mode_new=<-rw----->, attr_new=<----->,  
imode_new=<33152>, iattr_new=<0>, hardlinks_new=<1>, idevice_new=<0>,  
inode_new=<2752943>, owner_new=<root>, iowner_new=<0>,  
group_new=<root>, igroup_new=<0>, size_old=<0>, size_new=<15779>,  
ctime_new=<[2016-02-14T13:16:02]>, atime_new=<[2016-02-14T13:16:02]>,  
mtime_new=<[2016-02-14T13:16:02]>,  
chksum_new=<7BC4A8D898293FE2A4F0F2710810FD3A650A4DBF40B4F212>  
  
CRIT    : [2016-02-14T11:16:04-0200] msg=<POLICY MISSING>, path=</etc-  
/samhainrc~>  
  
CRIT    : [2016-02-14T11:16:04-0200] msg=<POLICY MISSING>,  
path=</etc/.samhainrc.swp>, mode_old=<-rw----->,  
attr_old=<----->, imode_old=<33152>, iattr_old=<0>, hard-  
links_old=<1>, idevice_old=<0>, inode_old=<2753365>, own-  
er_old=<root>, iowner_old=<0>, group_old=<root>, igroup_old=<0>,  
size_old=<4096>, size_new=<0>, ctime_old=<[2016-02-14T13:15:59]>,  
atime_old=<[2016-02-14T13:15:56]>, mtime_old=<[2016-02-14T13:15:59]>,  
chksum_old=<EE67A507BF1B7847FDD36B0E4B86C31390CA0E15F1E74F1F>
```

As you can see, Vim removed (POLICY MISSING) /etc/samhainrc and created it again (POLICY ADDED). Vim also worked with samhainrc~, samhainrc.swp and the mysterious '4913' file. To this day, I have not discovered what reason Vim uses the 4913 file, but it happens in different operating systems. When monitoring your systems using Samhain you start discovering the modus operandi of your major softwares, and in no time you know how a change was made only reading log files.

Some important considerations:

- If you are using the inotify feature, the results will be different than when you are not using it;
- Sometimes you can have bursts of messages, you will need to adjust your samhainrc until Samhain gives only the important information to you;
- It is very useful to review Samhain messages after an installation, or any other big change in your system.

When you are using some HIDS like Samhain, it is important to monitor it because someone can stop it to change the content of critical files. By default, Samhain sends a time stamp message to its log. The interval can be configured using `SetLoopTime`, and the default value 600, representing 10 minutes. It is recommended to make alerts in your infrastructure monitoring system when these messages stop coming for more than a specific period of time (e.g. 30 minutes). `TIMESTAMP` messages are similar to the following:

```
MARK      :   [2016-02-14T11:19:18-0200]  msg=<----  TIMESTAMP  ---->
```

Samhain can send messages to different destinations, including logfiles, prelude, and the Yule log server. One alternative to reach a reasonable level of security, without a small effort and complexity, is to send the messages to Syslog daemon. By default, Samhain sends messages to local2 syslog facility, as you can see:

```
SyslogFacility=LOG_LOCAL2
```

Make sure that the facility local2 of your syslog is configured to send messages to a remote place. Doing that, you will be able to investigate changes without the necessity of log in the system.

A much more secure way to make messages of Samhain secure is using Yule log server; this subject will not be covered in this article.

What will I monitor?

Until now, we have seen the basics of the functioning of the Samhain software. Let us discover how we can configure Samhain to monitor what we need it to do.

We have two basic directives to inform Samhain about what needs to be monitored: `file` and `dir`.

An example of `file` is:

```
file= /etc/passwd
```

Using `file`, Samhain will monitor the file `/etc/passwd`. Wildcard patterns (`'*`, `'?'`, `'[...]'`) as in shell globbing are supported for paths. The leading `'/'` is mandatory. The `dir` directive is a bit more complex and useful. You need to inform Samhain the directory depth and Samhain will monitor the files inside this directory until the depth is reached.

The maximum depth that Samhain accepts is 99 and we use it to inform Samhain that we need to monitor the entire content of the directory. The example of the use of dir is showed below:

```
dir = 99/etc
```

Samhain can monitor owner, group, permissions, file type, device number, hardlinks, links, inode, checksum, size, mtime, ctime, and atime. Posix ACLs and Selinux can also be monitored, but in this case it is necessary support comes from the operating system and you need to compile Samhain with these supports as well.

To simplify the configuration, Samhain has monitoring policies. Let us see the most important, and what they monitor:

ReadOnly: All modifications except access times (atime).

LogFiles: Modifications of timestamps, file size, and signature will be ignored.

GrowingLogFiles: Modifications of timestamps, and signature will be ignored. Modification of the file size will only be ignored if the file size has increased.

Attributes: The content of files is not monitored.

IgnoreAll: Monitor only if the file or directory exists or not.

IgnoreNone: Monitor all modifications.

User0 to User4: You can personalize your monitor policy.

Let us look at an example:

```
[ReadOnly]

dir = 99/etc


[Attributes]

file = /etc/mtab
```

In the configuration above, Samhain has been informed that it needs to check `/etc/`, and all the files and directories inside using the section `[ReadOnly]`. The operating system changes the file `/etc/mtab` frequently and because of this we need to create a specific configuration for it. The operating system changes only the content of the file and our intention is to monitor attributes of this file, such as the permission, owner, and group. As we saw above, we can use the 'Attributes' policy to monitor `/etc/mtab`. Samhain will use the most specific case before it evaluates what policy will be used, in this case `/etc/mtab` will match in Attributes policy instead of ReadOnly policy.

Sometimes you need to suppress messages of some specific action, like file or directory addition, modification, or deletion. You can use regular expressions in these directives. Let us look at a suitable configuration to monitor `/var/log` directory when we are using logrotate for syslog. Our `/var/log` directory will have auth and syslog files, with or without, the extension of `.gz`:

```
[GrowingLogFiles]

dir = 99/var/log


[Misc]

IgnoreAdded = /var/log/(syslog|auth) (\..log)?\..[0-9] (\..gz)?$

IgnoreMissing = /var/log/(syslog|auth) (\..log)?\..[0-9] (\..gz)?$
```

Logrotate can create and delete files like `/var/log/auth.log.1.gz` or `/var/log/syslog.2` without alerts. The content of the files can grow, but if it shrinks Samhain will give an alert. Similarly, if some attribute like mode, owner, or group change, Samhain will give an alert.

It's important to note that when you change what you are monitoring, you need to run the command `samhain -t update`. This command will update the baseline database that is the file

```
var/lib/samhain/samhain_file.
```

This is only the beginning

I hope you enjoyed this brief introduction to working with Samhain, as it is much more complex and useful than what we have seen in this article. Samhain can be installed in almost any POSIX system (*BSD, Linux, Solaris, AIX, HP-UX, and Mac OS X) and even in Windows systems using Cygwin. The server side, named Yule, can be installed only in POSIX systems.

Let us look at some features that were not covered in this article:

Centralized monitoring: Using Yule, Samhain can store baseline databases and client configurations in a centralized way. In this case, logs can be centralized and the log traffic will be encrypted.

Web-based management console: The Samhain project offers Beltane, which is a web-based console. It allows for the monitoring of server and client activities, a view of client reports, and an update of baseline databases.

Flexible Logging: Samhain supports multiple logging facilities, each of which can be configured individually.

Tamper Resistance: You can protect the baseline database and the configuration files using PGP.

Has many other modules: Samhain can detect rootkits and monitor suid/sgid binaries, login/logout events, mounted filesystems, hidden/fake/missing processes, open ports, and logfiles in UNIX systems. In windows systems, Samhain can monitor registry keys.

It can work with audit: It is possible to integrate Samhain with the audit subsystem to discover who made the change in the file or directory.

It is extensible: You can even create new modules for Samhain, if necessary, using C language.



About the Author:

Leonardo Neves Bernardo got started with UNIX in 1996, when he considered this operating system to be more interesting than any other system at that time. For more than twenty years he has worked in several IT areas, but has always been focused on UNIX operating systems.

Leonardo holds a Bachelor's degree in Computer Science from the Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina, Brazil, as well as various certifications including LPIC-3, LPIC-300, LPIC-302 and LPIC-303, RHCSA, and the ITILv3 Foundation. Visit his LinkedIn profile at: <https://br.linkedin.com/in/leonardoneves>

PRACTICAL PYTHON WORKSHOP

PEDRO ARAÚJO
& RUI SILVA

Module 1 (introduction):

Why python?

- Introduction about python programming language.
- Learning the strengths of the language and what's good with python.
- Learning where to use Python and why.
- Python as an interpreted language
- How to choose correct interpreter, install it, run it.
- Python virtual environments.
- Text editor (kate, gedit, brackets).
- How to create Hello world, from interpreter and with .py script.
- Standards and batteries included
- Standards and PEP8.
- Batteries included (just to show the most useful python core libraries and link to the official documentation).

Module 2 (python basics):

- Python data types and flow control statements
- Ifs, fors, whiles
- Lists (slices), dictionaries (loop over items), sets
- Python internals
- Classes and object instances
- Everything is an object (docs strings, getters, setters, override)
- Exceptions handling
- Practical example
- Use twitter's API to get some data and show it in the console

Module 3 (files):

- Files
- Duck typing.
- Opening and reading from files.
- Csv files and csvreader.
- Practical exercise

- Read file with a sentence per line.
- Manipulate and gather metrics on each sentence.
- Manipulate data to suit our needs.
- Plot a graph to show the data in a graphical and understandable way.

Output a file with metrics on each sentence.

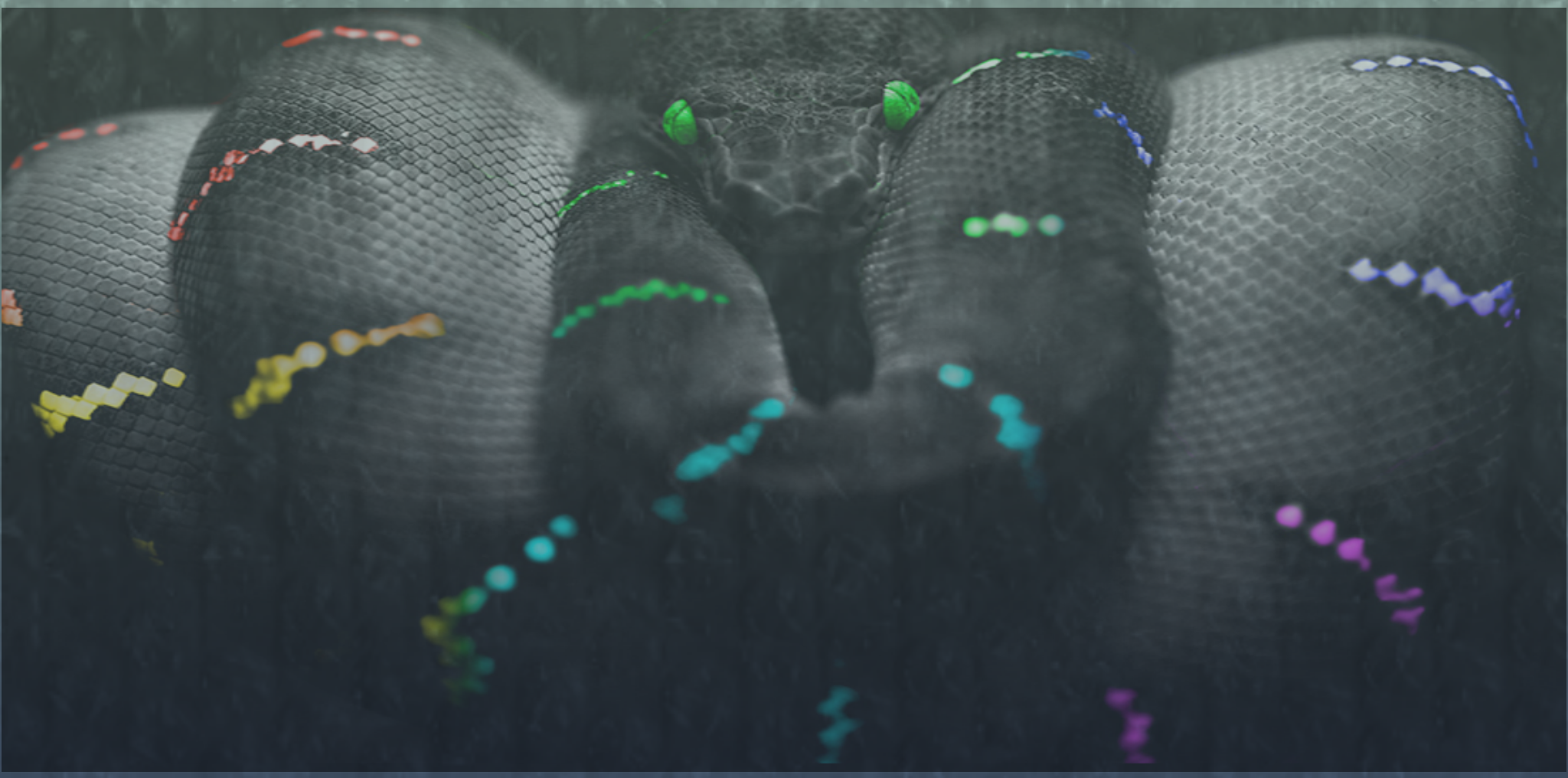
Module 4 (project):

- Practical project
- Get data from external source (<http://openweathermap.org>).

Authors

Pedro Araujo and Rui Silva

If you have any questions or just want to get to know us better feel free to contact me at marta.ziemianowicz@bsdmag.org or visit <https://bsdmag.org/course/python-programming-coming-next/>



If you have an idea, quickly take the plunge.

Micha Mazaheri

by Marta Ziemianowicz, Marta Sienicka & Marta Strzelec

[BSD Magazine]: Hello Micha, how have you been doing? Thank you for agreeing on the interview. Would you like to tell us something about yourself?

[Micha Mazaheri]: Hi! Thank you for having me. I started Paw almost 3 years ago as a side project. The idea was to build the tool I was wishing to have for myself when I was an iOS developer at an early stage [of a] San Francisco startup. Interestingly, I was not much focused on the business side for a long time, and I was spending time on probably unnecessary details in order to build the app that made me happy myself.

[BSD Mag]: You are a Founder of Paw. Can you tell our Readers something more about the company?

[MM]: We are tool builders. Not Iron Age tools nor machine tools, but productivity software for developers. While I started Paw with the goal of creating one specific Mac app, we are now aiming to build a complete workflow for teams working with APIs.

[BSD Mag]: Your product is also called Paw? What does it do?

[MM]: Paw is a developer tool for Mac that lets web API developers and consumers test and interact with HTTP servers: what is now commonly known as an HTTP client. Soon, it will become a SaaS service for team collaboration with a web frontend to test APIs and build their documentation seamlessly.

[BSD Mag]: What is the huge update you have been working on recently?

[MM]: Fun fact, it is our first update with a codename: Keep it secret, keep it safe. The name describes perfectly what it does; finally bringing privacy control and local encryption to our API testing tool. We are very excited about this update as Paw now becomes the only truly secure HTTP client, while our competitors are still sending their customers API credentials in clear to their servers.

[BSD Mag]: What is the strongest feature of Paw? Am I right it is security?

[MM]: Security has been our focused recently, but our secret sauce is clearly what we call dynamic values. This mechanism allows users to quickly parametrize the HTTP calls they compose in a very visual way; in some cases, one needs to fetch a particular field from another API, and compute a hash of it, combined along with a few headers of the current request. That is possible in Paw, without writing a single line of code.

[BSD Mag]: Are you an Open Source Software enthusiast?

[MM]: Enthusiasts we are! But not active enough yet to make us truly proud. We have many public projects on GitHub and are publishing all the JavaScript-based extensions we write for Paw under the MIT license. We have for example a bash script command line parser written in JavaScript ES6; though, many other projects are planned such as a browserless syntax highlighting library (based on Highlight.js) and a React component to go with it.

[BSD Mag]: How much Open Source is in Paw? What are you based on?

[MM]: The number of OSS projects we use is massive. Our Mac app uses the awesome CocoaAsyncSocket networking library, our backend is in Django, and our web frontend in React!

[BSD Mag]: Your company is a start-up. But you have some big clients already! How did you do this?

[MM]: To be honest, we have not done much marketing. I guess Paw is in a niche market, and our obsession with detail seems to have paid off.

[BSD Mag]: You are still working for another company, is that right? Do you have any tips for our Readers on how to make a start-up company successful?

[MM]: No, I am full time with Paw for about 18 months now, and we are now a team of 4. I consider ourselves still far away from being entitled to give any meaningful advice, but I would risk saying that if you have an idea, quickly take the plunge. Do not mind the legal aspects of creating a company at first - just do it. Learn from your mistakes, abandon a project quickly if your guts say it is the wrong one, and jump into another one. Carefully balance quick iteration and care of the details.

[BSD Mag]: Is there any story behind the company? Or a philosophy that drives you to success?

[MM]: We follow our instincts, and listen to users. I guess it is again the idea of balancing the hard truth of user feedback and analytics data with your own gut feeling.

[BSD Mag]: Any piece of advice for our readers?

[MM]: As users and consumers, continue to be nice to young companies and entrepreneurs. Their life is tough and they often are the ones who build the products we all love! At Paw, we were lucky enough to mostly have only positive feedback and always dealt with friendly and understanding users – I am so thankful to them. But I can tell how much it hurts when someone directly criticizes your product.



About Micha:

Micha has always had a passion for technology and was a self-learning programmer before he studied science, math and CS. He founded Paw to change the way developers test, discover and build web APIs. He has a deep passion for detail, and his dream is to build a company where the team goes on beach or ski retreats and with an office that looks like a playground.

Paw

Paw is a Paris based startup with a small team and great challenges to solve. Our mission is to build the tools we wish were already existing.

Model View Whatever - MVP by Mike Potel

by Damir Czernous

In the 1990s, software engineers attempt to unite two dominating UI designs: Forms and Controls, and MVC with the Application Model (AMVC). The result is that the Model View Presenter (MVP) structure emerges. It is difficult to determine the MVP inventor. It seems to be a collective work occasionally summarized by specific engineers. In that design cloud, two mainstreams start clashing. One sympathizes with Mike's Potel's way of thinking. The other one promotes Andy Bower and Blair McGlasha's point of view. Generally however, MVP is referenced via the Potel paper MVP, "Model-View-Presenter The Taligent Programming Model for C++ and Java".

Redefined view

All agree on considering a view as a form of widgets (e.g. some window with a few text fields and buttons). This is inspired by the Forms and Controls approach described in the previous paper "Model View Whatever - Forms and Controls influence". Thus, MVP structure redefines the view. The new view "sees" a UI in a same way as human beings. It is now more natural to compose views using widget frameworks. That change however,

leaves behind the view/controller separation known from MVC.

Commands handle user actions

Potel cautiously discusses the issues of the user actions. He believes that their only job is to hand over user requests to the presenter (Figure 1 step 1.2). The presenter handles user requests by editing the model via command and selection objects (Figure 1 steps 1.2.1-2). It seems that Potel considers user actions in the context of the model

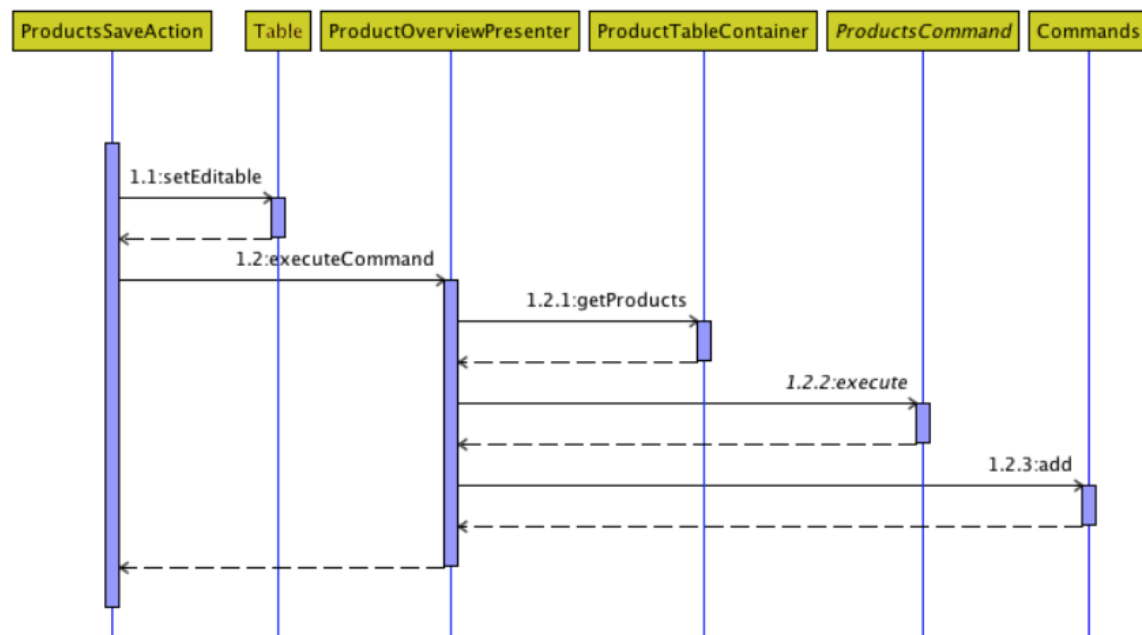


Figure 1: Potel's MVP handles user actions using commands

This leaves the management of the view states for other engineering considerations (Figure 1 step 1.1 can be done by an action, a view or a presenter). In more advanced UIs, the command object may provide undo and redo behaviour (see the example below).

```

public class ProductsSaveAction implements Button.ClickListener {
    private final ProductOverviewPresenter productOverviewPresenter; private
    ProductsSaveCommand productSaveCommand;
    private ProductTable productTable;

    @Override
    public void buttonClick( Button.ClickEvent clickEvent ) {

        productTable.setEditable( false );

        productOverviewPresenter.executeCommand( productSaveCommand );
    } }

    public class ProductOverviewPresenter {

        private final ProductOverviewView productOverviewView; private final
        Commands commands = new Commands();

        public void executeCommand( ProductsCommand productsCommand ) {

            Products products =
            productOverviewView.getProductTable().getContainer().getProducts();
            productsCommand.execute( products );
            commands.add( productsCommand );
        }
    }
    
```

```
public void undoCommand() { commands.undo(); }

public void redoCommand() { commands.redo(); } }

public class ProductsSaveCommand implements ProductsCommand {

private final ProductGateway productGateway; private Products undo-
Products;
private Products redoProducts;

@Override
public void execute( Products products ) {

undoProducts = productGateway.find(); redoProducts = products;
productGateway.save( products );

}

@Override
public void undo() { productGateway.save( undoProducts ); }

@Override
public void redo() { productGateway.save( redoProducts ); }

}
```

The view has no information on how the widgets handle user actions. It also uses an Observer Synchronization mechanism, known from MVC, to update displayed data.

MVP fabrication vs Object Oriented Design (OOD)

“The presenter then represents the traditional “main” or “event loop” part of the application,

creating the appropriate models, selections, commands, views, and interactors, and providing the business logic that directs what happens when, like a traffic cop or orchestra conductor.”

Mike Potel

```

public class ProductOverviewPresenter {

private final ProductOverviewView productOverviewView = new ProductO-
verviewView();

private final ProductGateway productGateway =
ProductGateway.create();

public void buildAndBind() {

ProductsSaveCommand productsSaveCmd = new ProductsSaveCommand( pro-
ductGateway );

    ProductsSaveAction productsSaveAction = new ProductsSaveAction( this
);

productsSaveAction.setCommandAndComponent( productsSaveCmd,
productOverviewView.getProductTable() );
productOverviewView.getSaveButton().addClickListener( productsSaveAc-
tion );

} }

```

Such fabrication abuses the patience of Separation of Concerns principle. Fabrication (red color) and building (purple color) are separate responsibilities that belong to the Main Pattern. The Main Pattern is the place where all factories and builders are implemented, but not the application logic. All dependencies

navigate toward application modules. That allows a customization of the applications by enabling a set of required functionalities. The code below does the exact same thing using the builder class and Main Patter approach.

```

productOverview.jar
package com.sanecoders.bakery.product.overview;

public class ProductOverviewPresenter {

private final ProductOverviewView productOverviewView; private final
ProductGateway productGateway;

public ProductOverviewPresenter( ProductOverviewView productOver-
viewView, ProductGateway productGateway ) {

```



```

this.productOverviewView = productOverviewView;

this.productGateway = productGateway; }

}

bakeryBuilders.jar // depends on productOverview.jar package
com.sanecoders.bakery.product.overview;

public class ProductOverviewBuilder {

private ProductOverviewView productOverviewView;
private ProductOverviewPresenter productOverviewPresenter; private
ProductGateway productGateway;
private ProductsSaveCommand productsSaveCmd;
private ProductsSaveAction productsSaveAction;

public void build() {

create();

buildProductSaveAction(); }

}

```

The presenter behaves like a façade as well. The facade can be quite useful when using MVW structures. It can not only create and wire (through factories and builders) parts of the MVP, but also simplifies its usage. This is a convenience way of using widgets served by different widget

frameworks that allow for editing styles and states through the single API; however, the facade is optional. Facades, for the propose of

ease of use, often violate one or more of the SOLID12 principles; therefore, it makes sense to hold them separately.

In the next paper

In 2000, in San Diego, Andy Bower and Blair McGlashan publish a paper “Twisting The Triad - The evolution of the Dolphin Smalltalk MVP application framework”. In contrast to Mike Potel, they pay more attention to the communication between the view and the presenter. Such delight is completely understandable for heirs of MVC. The next paper, “Model View Whatever - Dolphin Smalltalk MVP” describes the view/presenter communication in detail.

<https://www.linkedin.com/in/mikepotel>

<https://www.linkedin.com/in/andy-bower-5766ba3>

<https://www.linkedin.com/in/blair-mcglashan-01a4726>

<http://www.wildcrest.com/Potel/Portfolio/mvp.pdf>

<http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod>



About the Author:

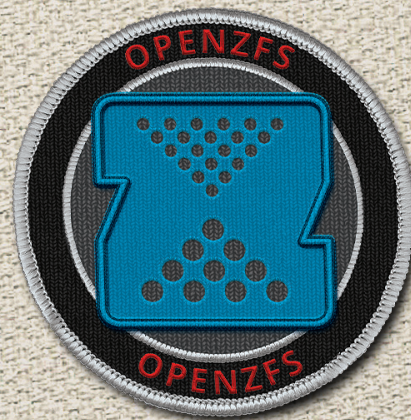
Damian Czernous

Reasoning about software architecture fascinates me for 10 years now.

Lead Engineering Coach at Nokia.

www.sanecoders.com, [@DamianCzernous](https://twitter.com/DamianCzernous)

#MISSIONCOMPLETE



"CryptoLocker is a joke with ZFS"

Learn how Plextec defeats ransomware attacks with
FreeNAS and ZFS at ixsystems.com/cryptolocker

Have you used one of these tools to complete your mission?

Tell us more at ixsystems.com/missioncomplete for a chance to win monthly



#missioncomplete



Solving Storage Challenges with Root on ZFS in FreeNAS and TrueNAS

by Mark VanFange

The ZFS file system provides data integrity features for storage drives using its Copy On Write (CoW) technology and improved RAID, but these features have been limited to storage drives previously. If you have a drive failure, utilizing RAID or mirroring will protect your volumes, but what happens if your boot drive fails?

In the past, if you used FreeNAS, you had no option other than having your storage go offline and remain unusable until it was repaired, and the ability to mirror was only available in TrueNAS, which utilized the underlying FreeBSD code.

In older versions, the FreeNAS and TrueNAS boot drives used the UFS (Unix File System), an older file system that does not include the advanced data integrity features found in ZFS. This has recently changed on current versions of TrueNAS and FreeNAS, and now ZFS can be installed on boot drives using the menu-driven installer via a simple interface.

The addition of Root on ZFS to FreeNAS and TrueNAS brings those data integrity features to the boot drives, providing users the ability to improve their storage units' reliability, and improve availability (meaning less downtime) by setting up their operating system drives in a mirror configuration.

Another improvement is the bootloader, which root on ZFS takes advantage of. The previous bootloader did not work well with multiple boot environments. As of the FreeNAS 9.3 release, FreeNAS and TrueNAS have moved over to the GRUB bootloader, which is much better equipped for this functionality.

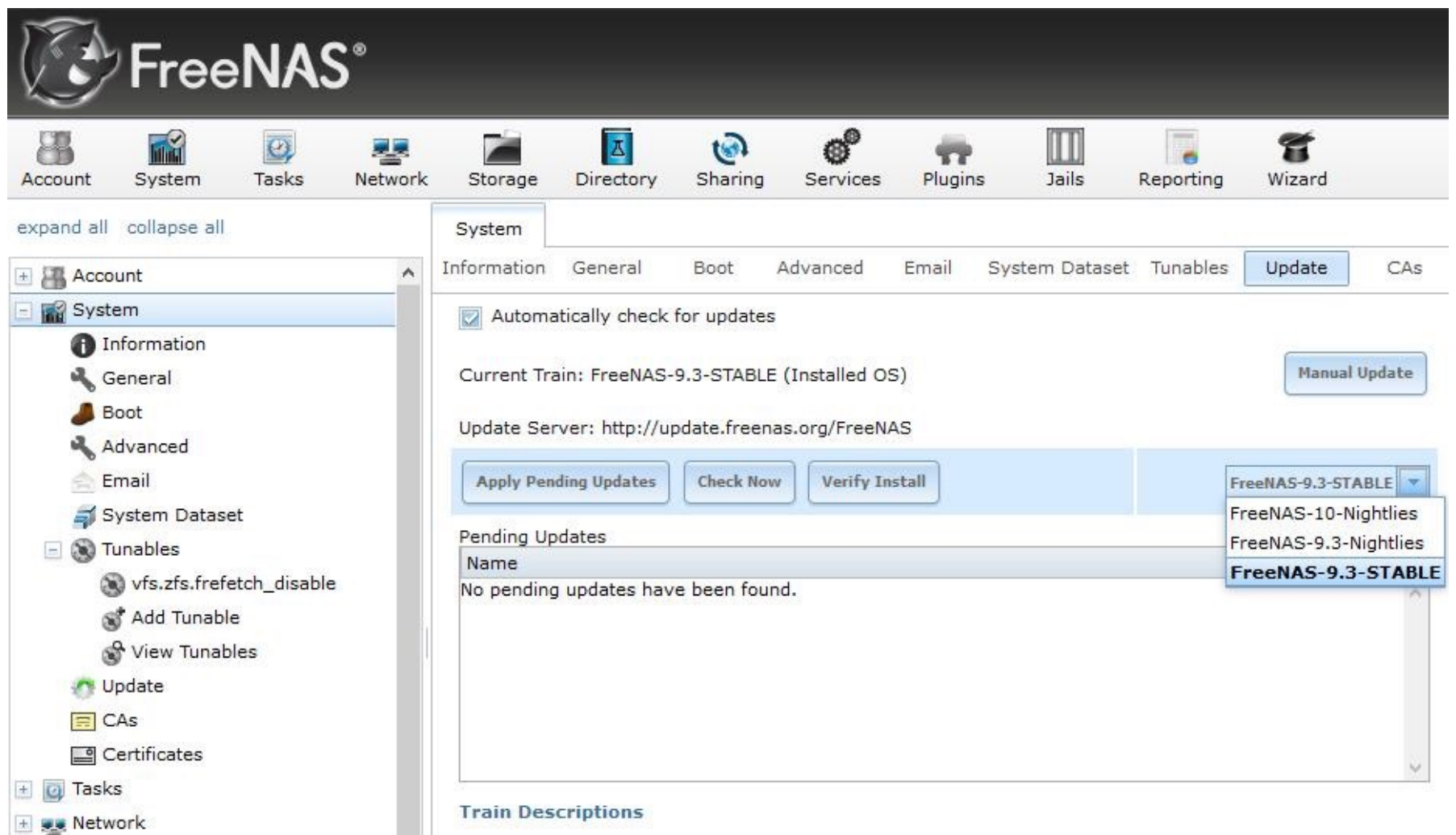


Image 1. The GRUB bootloader Update.

This update works hand in hand with the FreeNAS upgrade system, allowing users to switch between nightly and stable builds, as well as rollback to previous versions, with ease.

Conclusion

By incorporating Root on ZFS technology for boot drives TrueNAS and FreeNAS gain an improvement over previous boot technology by incorporating ZFS-based data integrity protections. Incorporating those protections into the boot drives improves reliability by detecting and repairing drive and volume errors. It

improves storage availability by eliminating downtime in the case of an OS drive failure and takes advantage of ZFS's self-healing capabilities to decrease downtime by detecting and, if mirrored, fixing errors. In addition, this change utilizes an improved bootloader, making operating system version upgrades and rollbacks run smoothly.

The FBI and Apple are engaged in very public spat concerning encryption, data privacy and intellectual property. Who should we be more afraid of – government, business or terrorists?

by Rob Somerville

I'll be honest. Normally when preparing a column, I will spend some time researching the pros and cons of the debate, and try and dig up some interesting facts that may help sway the reader towards my particular point of view or, at the very least, cast an interesting alternative light on a particular subject. Now that the Apple, FBI and the legal machinery PR is rotating at full speed, the chances of accurately measuring the gravity of the current situation will be buried beneath the spin. Hence, my reticence to research the minutiae of this current débâcle, and consequently be drawn down a rabbit hole of opinion, agenda, bias or political rhetoric. So this column is written purely from the perspective of a pragmatic, objective observer of life rather than somebody who has a particular axe to grind. In short, there will be something to offend everyone, whether you be government, business or a terrorist.

Morality and ethics are a strange arena. As life is not black and white, on a daily basis, as individuals, we have to make value judgments that may have serious repercussions on other individuals and society. I unequivocally believe that the vast majority of people want to do the right thing. The problem gets more complicated the further up the greasy pole of responsibility you climb. What may be right for you may not necessarily be right for your family, your community or indeed your country. Likewise, what may be right for your country may not be right for the individual. Traditionally, these differences of values are meant to be addressed via the ballot box, or the jury, or to quote the politicians rhetoric, "democracy". Unfortunately, in real life, we do not have a regular hot-line to our political leaders that allows us to plead our case as individuals when the shit hits the fan – that particular luxury only happens every four or five years, depending on if you live in the USA or UK. Sadly though, as a nation, we have very short memories. What happened mid-term is often forgotten amidst the carnival fever at election time, and the bulwark of historical legislation is used by the next new leader based on precedent. The cover on the book changes, but rarely the content.

And so we end up with a commonality that is so obscene, in reality, few wish to consider or even more rarely, confront it. The government, the business and the terrorist all have their underlying foundations based on power, and depending on the particular cycle, their respective covers are changed but the underlying philosophies often remain the same. Control, influence, social acceptance, but also recognition. Sincere contributions are made by dedicated individuals – some corrupt and some less corrupt than others.

Some paying with their wallets, some with family time, some with their lives. All of this blends into the "Big cause". I am right, you are wrong. We must draw the line somewhere. No room for understanding, negotiation, natural justice or peace. The individual will bow down and follow the system, irrespective of how rotten or corrupt it may be.

One of the biggest problems we have in society today is the pervasive erosion of values by the philosophy of situational ethics. You would think that having X thousand number of years on this planet (replace X with the value of your choice) we would have worked out a few ground rules – and decided to stick with them. No. We have entered the age of post-modernism, where right and wrong are movable posts depending on your point of view, the size of your bank account or your lobby group. Hence, our confusion these days about what is right and wrong as a society.

There is such irony and paradox in the current flame war between government, business and the terrorist. All have skeletons in their cupboards, yet all wish to claim the moral high ground. The millions of words typed via the Internet pro and con each argument - rather than help clarify the situation, they are just a distraction from the dark deeds and bad precedents of the past. We have totally lost focus of the real issues. Killing people is wrong. Allowing just and reasonable laws to be undermined and the pursuit of justice to be perverted is wrong. Ignoring the avarice of corporations who take a particular line not so much because of the benefits to society but for sheer profit alone and to hell with the consequences is wrong. Justice, and the consequences of breaking the law should be applied with equal force to all, be it the individual, the business, the government or the terrorist. Yet so often, decisions are taken and society deliberately kept in the dark for the very simple reason there would be hell to pay if the truth be known. And even then, as individuals we are often guilty of not taking a stand when we do see injustice for fear of rocking the boat or of being accused of being awkward.

If we as a society are to root out the destructive forces that threaten to destroy our freedoms, our values, our very way of life, we must first expose them. Currently, all we have is a giant finger pointing exercise where one side wants to be right and prove the other wrong, totally ignoring the fact that both are, to varying degrees, complicit – historically, at least. We desperately need less confusion, a return to the good old fashioned values of right and wrong. Whoever gains the upper hand in this particular scenario has a huge responsibility to carry - be it government or corporation - for if they get it wrong the consequences for us all don't bear thinking about. For once the precedent is set, it will be mercilessly leveraged and a small incident will, over time, amplify to become something much larger.

When it comes to who is the most discredited in this battle, only time will tell. All players, be it government, business or the terrorist at the moment, seem to be doing their damndest to win the prize of who reaches the bottom of the ethical barrel first.